Theoretical Computer Science 411 (2010) 2878–2889

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

The bridge-connectivity augmentation problem with a partition constraint

Yen-Chiu Chen [a], Hsin-Wen Wei [b,*], Pei-Chi Huang [a], Wei-Kuan Shih [a], Tsan-sheng Hsu [b]

[a] *Department of Computer Science, National Tsing-Hua University, No 101, Section 2, Kuang-Fu Road, Hsinchu, Taiwan*

[b] *Institute of Information Science, Academia Sinica, No 128, Section 2, Academia Road, Nankang, Taipei, Taiwan*

ARTICLE INFO

Article history:
Received 9 October 2009
Received in revised form 19 March 2010
Accepted 7 April 2010
Communicated by D.-Z. Du

Keywords:
2-edge-connectivity
Bridge-connectivity
Augmentation
Partition constraint

ABSTRACT

In this paper, we consider the augmentation problem of an undirected graph with k partitions of its vertices. The main issue is how to add a set of edges with the smallest possible cardinality so that the resulting graph is 2-edge-connected, i.e., bridge-connected, while maintaining the original partition constraint. To solve the problem, we propose a simple linear-time algorithm. To the best of our knowledge, the most efficient sequential algorithm runs in $O(n(m+n\log n)\log n)$ time. However, we show that it can also run in $O(\log n)$ parallel time on an EREW PRAM using a linear number of processors, where n is the number of vertices in the input graph. If a simple graph exists, our main algorithm ensures that it is as simple as possible.

1. Introduction

A graph is said to be *k-edge-connected* if it remains connected after the removal of any set of edges whose cardinality is less than k. Finding the smallest set of edges to make an undirected graph k-edge-connected is a fundamental problem in many important applications; readers may refer to [1–3] for a comprehensive survey.

Augmentation problems in bipartite graphs are studied in [4–6]. The related 2-edge-connectivity augmentation problem arises naturally from research on the security of statistical data [7–11]. The information is stored in a cross-tabulated table and it is common practice to suppress some of the cells in the table to protect sensitive information. A basic concerning issue of this practice is its effectiveness, that is how to suppress a small number of cells to protect the information, so that the resulting table does not leak important information and the sensitive information will not be revealed to an adversary. This protection problem can be reduced to an augmentation problem in bipartite graphs [12,6,13,14]. In addition, many algorithms have been developed to resolve the problem of making general graphs k-edge connected or k-vertex connected for various values of k [15,16,4,17–19]. For example, a linear-time algorithm for the smallest bridge-connectivity augmentation problem in a general graph that does not have a partition constraint is proposed in [15]; while a linear-time algorithm for the bridge-connectivity augmentation with a bipartite constraint is described in [20]. Jensen et al. [5] presented an algorithm that solves the k-edge-connectivity augmentation problem in a graph containing partition constraints in $O(n(m+n\log n)\log n)$ time, where n is the number of vertices, and m is the number of distinct edges in the input graph.

In this paper, we focus on augmenting graphs with a partition constraint. Specifically, the constraint requires that the vertex set of an input graph must be partitioned into k disjoint vertex subsets, and each edge in the augmentation must be added between two different vertex subsets. We propose a linear-time algorithm that addresses the problem of adding the smallest number of edges to a graph with a given partition constraint to make it 2-edge-connected, or bridge-connected, while maintaining the constraint. Fig. 1(a) shows an example of a graph with three partitions of the vertices. A smallest

[*] Corresponding author. Tel.: +886 2 2788 3799x2471; fax: +886 2 2782 4814.
E-mail addresses: ycchen@rtlab.cs.nthu.edu.tw (Y.-C. Chen), hwwei@iis.sinica.edu.tw (H.-W. Wei), peggy@rtlab.cs.nthu.edu.tw (P.-C. Huang), wshih@cs.nthu.edu.tw (W.-K. Shih), tshsu@iis.sinica.edu.tw (T.-s. Hsu).

0304-3975/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2010.04.019

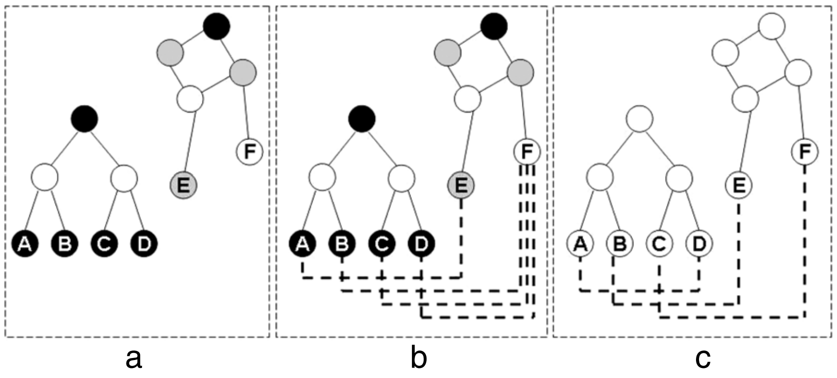

Fig. 1. (a) A graph with three partitions of the vertices. (b) A smallest 2-edge-connectivity augmentation of (a) with the set of added edges marked by dashed lines. (c) A smallest 2-edge-connectivity augmentation of (a) without a partition constraint where the set of added edges are marked by dashed lines.

2-edge-connectivity augmentation of the graph in Fig. 1(a) is shown in Fig. 1(b). Moreover, a smallest 2-edge-connectivity augmentation of (a) without a partition constraint is shown in Fig. 1(c). Note that there are 3 added edges in Fig. 1(c), however, there are 4 added edges in Fig. 1(b) which should satisfy a partition constraint. It is obvious that the set of added edges in Fig. 1(c) is not valid for Fig. 1(a) since vertices A and D are in the same partition in Fig. 1(a).

We solve the problem of finding a smallest 2-edge-connectivity augmentation of graphs with a partition constraint by transforming the input graph G into a well-known data structure called a *bridge-block forest* [21]. Our approach adds the smallest possible number of edges to make a bridge-block forest 2-edge-connected. The edge set added to the bridge-block forest by our algorithm can be transformed into the corresponding edge set added to the input graph G. The proposed algorithm runs in sequential linear time or $O(\log n)$ parallel time on an EREW PRAM using a linear number of processors.

The remainder of this paper is organized as follows. Section 2 provides detailed graph-theoretical definitions and previously known properties. In Section 3, we introduce the concept of edge swapping and the properties of the swap function. In Section 4, we propose an algorithm that finds a smallest 2-edge-connectivity augmentation for a bridge-block forest. Section 5 contains some concluding remarks.

2. Preliminaries

2.1. Graph-theoretical definitions

Let a graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. G is a *tree* if it is an undirected graph that is also connected and acyclic; and a maximal connected subgraph is a *component* of G. A *forest* is a graph whose components are all trees, and a degree-1 vertex of a forest is called a *leaf*. An edge whose endpoints are a vertex u and a vertex v is denoted as (u, v). Note that for an edge set E', $G - E'$ denotes G without the edges in E', and $G \cup E'$ denotes G with the edges in E' added to it.

In this paper, all graphs are undirected, and have neither self-loops nor multiple edges. The vertex set of an input graph is assumed to be partitioned into k disjoint partitions, called *vertex partitions*. Let P_i denote the set of vertices in the i-th vertex partition, i.e., P_i is a subset of V and $P_i \cap P_j = \emptyset$, $i \neq j$; and let $V = P_1 \cup P_2 \cup \cdots \cup P_k$. In addition, let $e = (u, v)$ be an edge with two endpoints, u and v, in different vertex partitions of an input graph, i.e., $u \in P_i$, $v \in P_j$, $i \neq j$. We call such an edge a *legal edge*. Our problem is how to add a set of edges such that the resulting graph is 2-edge-connected and all added edges are legal.

2.2. Bridge-block forest

A vertex u is *connected* to a vertex v in a graph G if u and v are in the same connected component of G. Two vertices of a graph are 2-*edge-connected* if they are in the same connected component and remain so after the removal of any edge. A set of vertices is 2-*edge-connected* if each pair of its vertices is 2-edge-connected; similarly, a graph is 2-*edge-connected* if its set of vertices is 2-edge-connected. A *bridge* is an edge in a graph G, the removal of which would increase the number of connected components of G by one. Given a graph G with at least three vertices, a smallest 2-*edge-connectivity augmentation* of G, denoted by aug2$e(G)$, is a set of edges with the minimum cardinality whose addition would make G a 2-edge-connected graph if it exists.

A *block* in a graph is an induced subgraph of the maximal 2-edge-connected subset of vertices. If a block contains all the nodes in a connected component of G, it is called an *isolated block*. The *bridge-block graph* of an undirected graph G, denoted by BB(G), is defined as follows. Each block is represented by a vertex of BB(G). When all the blocks in G are represented by vertices, BB(G) becomes a forest, such that each bridge in G corresponds to an edge in BB(G) and vice versa. For example,

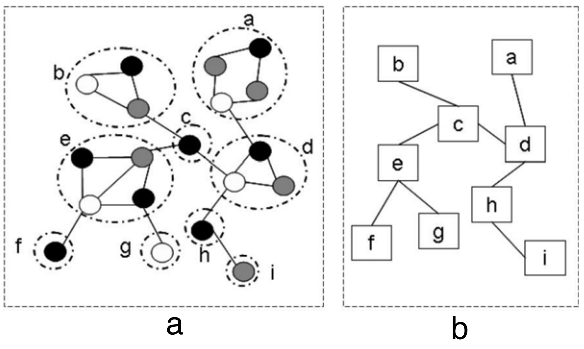

Fig. 2. (a) A graph with three vertex partitions; the maximum 2-edge-connected subsets of vertices of the graph are grouped into the set of blocks marked by the dashed lines. (b) The bridge-block forest of the graph in (a).

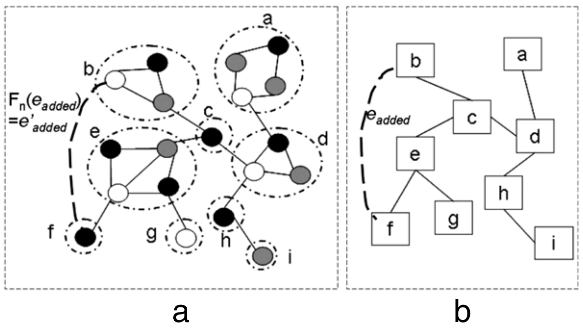

Fig. 3. Illustration of the transformation by F_n.

the blocks $a, b, \dots, i$ are represented by vertices. The resulting tree is illustrated in Fig. 2. In G, a *mono block of* P_i is a block comprised of the vertices in P_i ; and a *hybrid block* in G is a block containing at least two vertices, one in P_i and another in P_j, where $i \neq j$ and $P_i, P_j \in V$. A *mono isolated block of* P_i in G is both a mono block and an isolated block of P_i; and a *hybrid isolated block* in G is both a hybrid block and an isolated block.

The vertices and leaves in BB(G) are defined as follows. Given a graph G with k vertex partitions $P_1, \dots, P_k$, $k \geq 1$, a *mono leaf* (respectively, *mono vertex*) of P_i in BB(G) is a leaf (respectively, vertex) whose corresponding block in G is a mono block of P_i. A *hybrid leaf* (respectively, *hybrid vertex*) in BB(G) is a leaf (respectively, vertex) whose corresponding block in G is a hybrid block. A *mono isolated vertex of* P_i in BB(G) is a mono isolated block of P_i or an isolated vertex of P_i in G; and a *hybrid isolated vertex* in BB(G) is a hybrid isolated block in G. In addition, let F_n be a mapping that can transform an edge set added to BB(G) into the corresponding edge set added to G. If E' is the edge set added to BB(G), then $F_n(E')$ is the corresponding edge set added to G, i.e., aug2$e(G) = F_n(E')$. Similarly, if e' is an edge added to BB(G), then $F_n(e')$ is the corresponding edge added between a vertex u and a non-adjacent vertex v, where $u, v \in V$, $u \in P_i$, $v \in P_j$, and $i \neq j$. To take Fig. 2(a) as an example, a corresponding bridge-block forest is shown in Fig. 2(b). We assume that e_{added} is added between vertex b and vertex f which is shown in Fig. 3(b). Then, F_n can transform the added edge into a corresponding edge e'_{added}, i.e., $F_n(e_{added})$. The latter is added between a white vertex of block b and a black vertex of block f which is shown in Fig. 3(a).

Given a PRAM model M, let $T_M(n, m)$ be the parallel time needed to find the connected components of G using $P_M(n, m) \leq (n + m)$ processors.

Fact 1 ([22,23]). 1. *If $M = CRCW$, then $T_{CRCW}(n, m) = O(\log n)$ and $P_{CRCW}(n, m) = O((n + m)\cdot\alpha(m, n)/\log n)$.*
2. *If $M = EREW$, then $T_{EREW}(n, m) = O(\log n)$ and $P_{EREW}(n, m) = O(n + m)$.*

A rooted bridge-block forest for a graph can be computed in sequential linear time and in $O(\log n + T_M(n, m))$ parallel time using $O((n + m)/\log n + P_M(n, m))$ processors on an M PRAM [24–26].

Fact 2. *An edge can be added between two blocks in G, unless both blocks are mono blocks of a given partition P_i in G.*

3. Edge swapping

To reduce the complexity of finding a smallest augmentation of a graph, we define the following *swap* function. We use the following notations to illustrate the connectivity relationships in a graph: $u \overset{G}{\rightsquigarrow} v$ denotes a path in a graph G from a vertex u to a vertex v; $u \leftrightarrow v$ denotes the edge between u and v, i.e., the edge (u, v); and END(E_x) denotes the set of vertices that are endpoints of edges in the given edge set E_x.

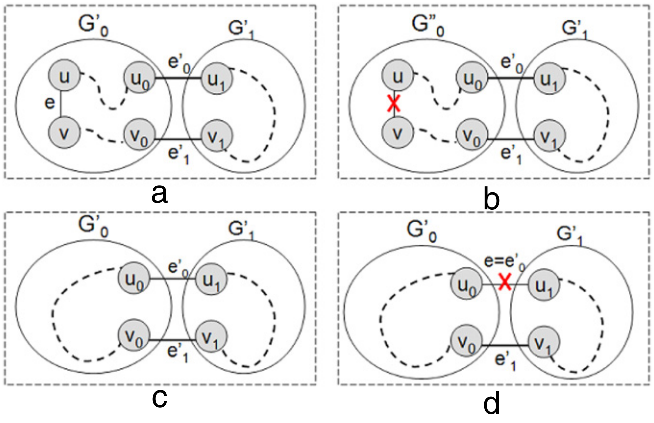

Fig. 4. (a) e is in G_0'. (b) $G_0'' = G_0' - \{e\}$. (c) G_0' and G_1'. (d) $e = e_0' = (u_0, u_1)$.

Definition 1. swap(e_0, e_1).
Let swap(e_0, e_1) be a function that takes a set of legal edges $E_{\text{in}} = \{e_0, e_1\}$ as an input and outputs a set of legal edges $E_{\text{out}} = \{e_0', e_1'\}$, where $e_0 = (u_0, v_0)$ and $e_1 = (u_1, v_1)$ are two legal edges, $\text{END}(e_0) \cap \text{END}(e_1) = \emptyset$ and $\text{END}(e_0') \cap \text{END}(e_1') = \emptyset$. Then, E_{in} and E_{out} satisfy the following properties: (1) $|E_{\text{in}}| = |E_{\text{out}}|$; (2) $\text{END}(E_{\text{in}}) = \text{END}(E_{\text{out}})$; (3) $E_{\text{in}} \cap E_{\text{out}} = \emptyset$; (4) the graph $G' = (\text{END}(E_{\text{in}}), E_{\text{in}} \cup E_{\text{out}})$ is a simple cycle; and (5) $\{e_0', e_1'\}$ is $\{(u_0, u_1), (v_0, v_1)\}$ or $\{(u_0, v_1), (u_1, v_0)\}$.

The following lemma describes the property of swap(e_0, e_1).

Lemma 1. *Given two legal edges, e_0 and e_1, which satisfy* $\text{END}(e_0) \cap \text{END}(e_1) = \emptyset$, swap$(e_0, e_1)$ *always exists.*

Proof. Let $e_0 = (u_0, v_0)$, and $e_1 = (u_1, v_1)$. We show that swap$(e_0, e_1) = \{e_0', e_1'\}$ is either $\{(u_0, u_1), (v_0, v_1)\}$ or $\{(u_0, v_1), (u_1, v_0)\}$, and that $G' = (\{u_0, u_1, v_0, v_1\}, \{e_0, e_1, e_0', e_1'\})$ is a simple cycle if swap(e_0, e_1) is $\{(u_0, u_1), (v_0, v_1)\}$ or $\{(u_0, v_1), (u_1, v_0)\}$. It is obvious that $|E_{\text{in}}| = |E_{\text{out}}|$, $E_{\text{in}} \cap E_{\text{out}} = \emptyset$, and $\text{END}(E_{\text{in}}) = \text{END}(E_{\text{out}})$.

Since (u_0, v_0) is a legal edge, u_0 and v_0 do not belong to the same vertex partition; similarly, u_1 and v_1 do not belong to the same vertex partition. Let P_1 and P_2 be two vertex partitions. Without loss of generality, we assume that $u_0 \in P_1$ and $v_0 \in P_2$, where $P_1 \neq P_2$. If $u_1 \in P_1$ or $v_1 \in P_2$, then swap(e_0, e_1) is $\{(u_0, v_1), (u_1, v_0)\}$, since $v_0 \notin P_1$ and u_0, v_1 do not belong to the same vertex partition. Hence, (u_0, v_1) and (u_1, v_0) are legal edges. In this case, G' is a simple cycle, i.e., $u_0 \leftrightarrow v_1 \leftrightarrow u_1 \leftrightarrow v_0 \leftrightarrow u_0$. Otherwise, swap$(e_0, e_1)$ is $\{(u_0, u_1), (v_0, v_1)\}$; however, because u_0 and u_1 do not belong to the same vertex partition, and neither do v_0 and v_1; thus, (u_0, u_1) and (v_0, v_1) are legal edges. Here, G' is a simple cycle, i.e., $u_0 \leftrightarrow u_1 \leftrightarrow v_1 \leftrightarrow v_0 \leftrightarrow u_0$. Therefore, from the above argument, swap(e_0, e_1) exists. □

Theorem 1. *Given two 2-edge-connected components G_0 and G_1 and two legal edges e_0 and e_1, let $G' = (G_0 \cup G_1 - \{e_0, e_1\}) \cup$ swap(e_0, e_1), where $e_0 \in G_0$, and $e_1 \in G_1$. Then, G' must be a 2-edge-connected component.*

Proof. To prove by contradiction, we let $e_i = (u_i, v_i)$ and assume that there exists a bridge $e = (u, v)$ in G'. That is, there will not be a path from u to v after removing e. Let swap$(e_0, e_1) = \{e_0', e_1'\}$, $G_0' = G_0 - \{e_0\}$ and $G_1' = G_1 - \{e_1\}$, then $G' = (G_0 \cup G_1 - \{e_0, e_1\}) \cup \{e_0', e_1'\} = G_0' \cup G_1' \cup \{e_0', e_1'\}$. According to Lemma 1, $\{e_0', e_1'\}$ is either $\{(u_0, u_1), (v_0, v_1)\}$ or $\{(u_0, v_1), (u_1, v_0)\}$. Without loss of generality, we assume that $\{e_0', e_1'\}$ is $\{(u_0, u_1), (v_0, v_1)\}$. We consider the following two cases.

Case 1. $e \in G_i'$, where $0 \leq i \leq 1$. Without loss of generality, we assume that $e \in G_0'$ and let $G_0'' = G_0' - \{e\}$. Note that G_0 and G_1 are 2-edge-connected components, there exist a cycle $u \overset{G_0}{\leadsto} u_0 \leftrightarrow v_0 \overset{G_0}{\leadsto} v \leftrightarrow u$ and paths $u \overset{G_0'}{\leadsto} u_0$, $v \overset{G_0'}{\leadsto} v_0$, and $u_1 \overset{G_1'}{\leadsto} v_1$. Since G_0 and G_1 are 2-edge-connected components and $G_0'' = G_0' - \{e\}$, there still exist $u \overset{G_0'}{\leadsto} u_0$ and $v \overset{G_0'}{\leadsto} v_0$. The difference of G_0' and G_0'' is shown in Fig. 4(a) and (b), respectively. Therefore, after removing e in G_0', paths still exist from u to v in G' through $u \overset{G_0''}{\leadsto} u_0 \leftrightarrow u_1 \overset{G_1'}{\leadsto} v_1 \leftrightarrow v_0 \overset{G_0''}{\leadsto} v$ which is shown in Fig. 4(b). Hence, G' is connected and e is not a bridge in G'.

Case 2. $e \in \{e_0', e_1'\}$. Since G_0 and G_1 are 2-edge-connected components, there exist paths $u_0 \overset{G_0'}{\leadsto} v_0$ and $u_1 \overset{G_1'}{\leadsto} v_1$ which are shown in Fig. 4(c). Without loss of generality, we assume that $e = e_0' = (u_0, u_1)$. After removing e, a path still exists from u_0 to v_1, i.e., $u_0 \overset{G_0'}{\leadsto} v_0 \leftrightarrow v_1 \overset{G_1'}{\leadsto} u_1$ which is shown in Fig. 4(d). In other words, G' is connected and e is not a bridge in G'.

To summarize, in both cases, there is no bridge in G' and G' is connected. Hence, G' is a 2-edge-connected component. □

Based on the properties of the swap function, we define a generalized function called *circular-swap*. Given a set of 2-edge-connected components, we select an edge from each component. Then, we apply the circular-swap function to the chosen set of edges E' and prove that the resulting graph, obtained by substituting E' with circularly swapped edges, is 2-edge-connected.

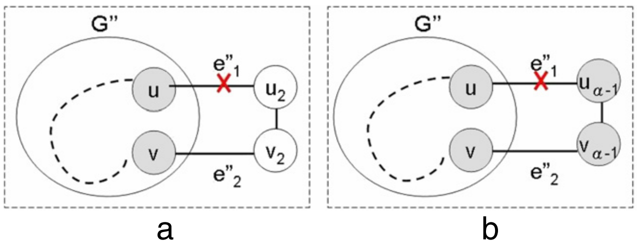

Fig. 5. Circular-swap$(e_0, \ldots, e_{h-1})$ (a) when $h = 3$. (b) when $h = \alpha$.

Definition 2. circular-swap$(e_0, \ldots, e_{h-1})$.
Let circular-swap$(e_0, \ldots, e_{h-1})$ be a function that takes a set of legal edges $E_{\text{in}} = \{e_0, \ldots, e_{h-1}\}$ as an input and outputs a set of legal edges $E_{\text{out}} = \{e'_0, \ldots, e'_{h-1}\}$, where $e_i = (u_i, v_i)$ is a legal edge, $\text{END}(e_i) \cap \text{END}(e_j) = \emptyset$ and $\text{END}(e'_i) \cap \text{END}(e'_j) = \emptyset$ for all i, j s.t., $0 \le i, j \le h-1$, $i \ne j$ and $e_i, e_j \in E_{\text{in}}$. Then, E_{in} and E_{out} satisfy the following properties: (1) $|E_{\text{in}}| = |E_{\text{out}}|$; (2) $E_{\text{in}} \cap E_{\text{out}} = \emptyset$; (3) $\text{END}(E_{\text{in}}) = \text{END}(E_{\text{out}})$; (4) the graph $G' = (\text{END}(E_{\text{in}}), E_{\text{in}} \cup E_{\text{out}})$ is a simple cycle; and (5) $e'_i = (u_p, u_q)$ or (u_p, v_q) or (v_p, v_q), where $0 \le i, p, q \le h-1$ and $p \ne q$.

Fact 3. *If E^* is an output for circular-swap(e_0, e_1), then E^* is also a possible output for* swap(e_0, e_1), *and vice versa, where* $\text{END}(e_0) \cap \text{END}(e_1) = \emptyset$.

For the readability, we let $x \longleftarrow\!\!\dashv y$ denote a solution for y is a solution for a circular-swap x in the remainder of this paper, where y is an operation on circular-swap functions.

Lemma 2. *Given a set of legal edges $E_{\text{in}} = \{e_0, \ldots, e_{h-1}\}$ as an input,* circular-swap$(e_0, \ldots, e_{h-1}) \longleftarrow\!\!\dashv$ (circular-swap$(e_0, \ldots, e_{h-2}) - \{e'_i\}) \cup$ swap(e'_i, e_{h-1}) *if $h > 2$, where $e'_i \in$* circular-swap$(e_0, \ldots, e_{h-2})$, $\text{END}(e_i) \cap \text{END}(e_j) = \emptyset$ *and* $\text{END}(e'_i) \cap \text{END}(e'_j) = \emptyset$, *for all $i, j, e_i, e_j \in E_{\text{in}}$, $0 \le i, j \le h-2$ and $i \ne j$.*

Proof. There are h edges in E_{in}. Let $e_i = (u_i, v_i)$ be a legal edge, $E_j = \{e_0, \ldots, e_{j-1}\}$ be the set of the input edges, $E'_j =$ circular-swap$(e_0, \ldots, e_{j-1}) = \{e'_0, \ldots, e'_{j-1}\}$ be the set of the output edges, and $G_j = (\text{END}(E_j), E_j \cup E'_j)$, where $0 \le i \le h-1$ and $2 \le j \le h$. In addition, let swap$(e'_i, e_{h-1}) = \{e''_1, e''_2\}$ and $\text{END}(e'_i) = \{u, v\}$, and $G' = (G_{h-1} \cup \{e_{h-1}\} - \{e'_i\}) \cup$ swap(e'_i, e_{h-1}). Without loss of generality, we assume that $e''_1 = (u, u_{h-1})$ and $e''_2 = (v, v_{h-1})$. We prove the lemma by induction on h.
Inductive basis: If $h = 3$, circular-swap$(e_0, e_1, e_2) \longleftarrow\!\!\dashv$ (circular-swap$(e_0, e_1) - \{e'_i\}) \cup$ swap(e'_i, e_2), where $e'_i \in$ circular-swap(e_0, e_1). Let $E_2 = \{e_0, e_1\}$ and $E'_2 = \{e'_0, e'_1\}$, the graph $G_2 = (\text{END}(E_2), E_2 \cup E'_2)$. Without loss of generality, we assume that $e'_i = e'_1 = (u, v)$ and $G'' = G_2 - \{e'_1\}$. We prove that $G' = G'' \cup \{e_2, e''_1, e''_2\}$ is a simple cycle as follows. By Fact 3, circular-swap$(e_0, e_1) \longleftarrow\!\!\dashv$ swap(e_0, e_1), i.e., circular-swap$(e_0, e_1, e_2) \longleftarrow\!\!\dashv$ (swap$(e_0, e_1) - \{e'_1\}) \cup$ swap(e'_1, e_2). After removing $e''_1 = (u, u_2)$, there still exists a path that is $u \overset{G''}{\rightsquigarrow} v \leftrightarrow v_2 \leftrightarrow u_2$ in $G' - \{e''_1\}$ and shown in Fig. 5(a). That is, G' is a simple cycle. Hence, the result holds.
Inductive step: Assume that the result holds for $h = \alpha - 1$, where α is a positive integer and $\alpha > 3$, i.e., circular-swap$(e_0, \ldots, e_{\alpha-2}) \longleftarrow\!\!\dashv$ (circular-swap$(e_0, \ldots, e_{\alpha-3}) - \{e'_i\}) \cup$ swap$(e'_i, e_{\alpha-2})$ exists, where $e'_i \in$ circular-swap$(e_0, \ldots, e_{\alpha-3})$.

For $h = \alpha$, we prove that circular-swap$(e_0, \ldots, e_{\alpha-1}) \longleftarrow\!\!\dashv$ (circular-swap$(e_0, \ldots, e_{\alpha-2}) - \{e'_i\}) \cup$ swap$(e'_i, e_{\alpha-1})$, where $e'_i \in$ circular-swap$(e_0, \ldots, e_{\alpha-2})$. Let $G_{\alpha-1} = (\text{END}(E_{\alpha-1}), E_{\alpha-1} \cup E'_{\alpha-1})$. Without loss of generality, we assume that $G'' = G_{\alpha-1} - \{e'_i\}$ and $e'_i = (u, v)$. We prove that $G' = G'' \cup \{e_{\alpha-1}, e''_1, e''_2\}$ is a simple cycle as follows. After removing e''_1, there still exists a path that is $u \overset{G''}{\rightsquigarrow} v \leftrightarrow v_{\alpha-1} \leftrightarrow u_{\alpha-1}$ in $G' - \{e''_1\}$ and shown in Fig. 5(b). That is, G' is a simple cycle.

Moreover, it is obvious that $|E_h| = |E'_h|$, $\text{END}(E_h) = \text{END}(E'_h)$, and $E_h \cap E'_h = \emptyset$ for any h, s.t., $h > 2$. Hence, the result holds. □

Lemma 3. *Given a set of legal edges $E_{\text{in}} = \{e_0, \ldots, e_{h-1}\}$ as an input,* circular-swap$(e_0, \ldots, e_{h-1})$ *exists if both* circular-swap$(e_0, \ldots, e_\ell)$ *and* circular-swap$(e_{\ell+1}, \ldots, e_{h-1})$ *exist for any ℓ, s.t., $0 < \ell < h-2$ and $h > 3$.*

Proof. Let circular-swap$(e_0, \ldots, e_{h-1}) \longleftarrow\!\!\dashv$ (circular-swap$(e_0, \ldots, e_\ell) \cup$ circular-swap$(e_{\ell+1}, \ldots, e_{h-1}) - \{e'_i, e'_j\}) \cup$ swap(e'_i, e'_j) and swap$(e'_i, e'_j) = \{e''_i, e''_j\}$, where $e'_i = (u_p, v_q)$, $e'_j = (u_{p'}, v_{q'})$, $0 \le i, p, q \le \ell$, $\ell + 1 \le j, p', q' \le h-1$, $p \ne q$, and $p' \ne q'$. Without loss of generality, we assume that $e''_i = (u_p, u_{p'})$ and $e''_j = (v_q, v_{q'})$. Let $E^1_{\text{in}} = \{e_0, \ldots, e_\ell\}$ be the set of the input edges and $E^1_{\text{out}} = \{e'_0, \ldots, e'_\ell\}$ be the set of output edges for circular-swap$(e_0, \ldots, e_\ell)$, then the graph $G_1 = (\text{END}(E^1_{\text{in}}), E^1_{\text{in}} \cup E^1_{\text{out}})$ is a simple cycle by Definition 2. Similarly, let $E^2_{\text{in}} = \{e_{\ell+1}, \ldots, e_{h-1}\}$ and $E^2_{\text{out}} = \{e'_{\ell+1}, \ldots, e'_{h-1}\}$ be the input and output edge sets for circular-swap$(e_{\ell+1}, \ldots, e_{h-1})$, respectively. Then the graph $G_2 = (\text{END}(E^2_{\text{in}}), E^2_{\text{in}} \cup E^2_{\text{out}})$ is a simple cycle.

Let E_{in} and E_{out} be the input and output edge sets for circular-swap$(e_0, \ldots, e_{h-1})$, respectively. Then, it is obvious that $E_{\text{in}} = E^1_{\text{in}} \cup E^2_{\text{in}}$, $E_{\text{out}} = (E^1_{\text{out}} \cup E^2_{\text{out}} - \{e'_i, e'_j\}) \cup$ swap(e'_i, e'_j), and the graph $G = (\text{END}(E_{\text{in}}), E_{\text{in}} \cup E_{\text{out}})$. Let $G'_1 = G_1 - \{e'_i\}$ and

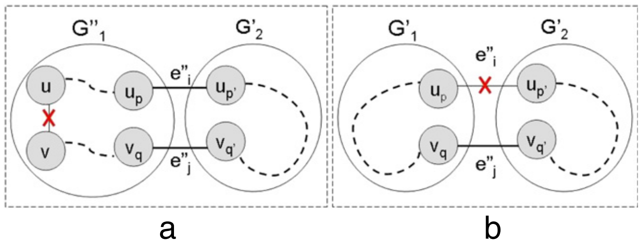

Fig. 6. (a) $G''_1 = G'_1 - e$, where $G'_1 = G_1 - e'_i$ and e is in G_1. (b) $e = e''_i = (u_p, u_{p'})$.

$G'_2 = G_2 - \{e'_j\}$. Since G_1 and G_2 are simple cycles, after removing e'_i from G_1 and e'_j from G_2, the paths $u_p \overset{G'_1}{\leadsto} v_q$ and $u_{p'} \overset{G'_2}{\leadsto} v_{q'}$ still exist.

It is clear to see that E_{in} and E_{out} satisfy the following properties: (1) $|E_{\text{in}}| = |E_{\text{out}}|$; (2) $\text{END}(E_{\text{in}}) = \text{END}(E_{\text{out}})$; (3) $E_{\text{in}} \cap E_{\text{out}} = \emptyset$; and (4) $e'_i = (u_p, u_q)$ or (u_p, v_q) or (v_p, v_q), where $0 \le i, p, q \le h-1$ and $p \ne q$. Therefore, assume that circular-swap$(e_0, \ldots, e_{h-1})$ does not exist, then G is not a simple cycle.

A graph with n vertices and n edges is a simple cycle if and only if it is connected and there is no bridge. First, we prove that G is connected, since a path $u_p \overset{G'_1}{\leadsto} v_q \leftrightarrow v_{q'} \overset{G'_2}{\leadsto} u_{p'} \leftrightarrow u_p$ exists; then we prove that there is no bridge. Assume there exists a bridge $e = (u, v)$ in the graph G. There are two possible cases to consider.

Case 1: $e \in G_{i^*}$, where $1 \le i^* \le 2$; that is, the bridge e is in G_1 or G_2. Without loss of generality, we assume that the bridge e is in G_1. Let $G''_1 = G'_1 - \{e\}$. Since G_1 and G_2 are simple cycles, there exist a cycle $u \overset{G'_1}{\leadsto} u_p \leftrightarrow v_q \overset{G'_1}{\leadsto} v \leftrightarrow u$ and the paths $u_p \overset{G'_1}{\leadsto} v_q$ and $u_{p'} \overset{G'_2}{\leadsto} v_{q'}$. Hence, after removing e, there is still a path in G that is $u \overset{G''_1}{\leadsto} u_p \leftrightarrow u_{p'} \overset{G'_2}{\leadsto} v_{q'} \leftrightarrow v_q \overset{G''_1}{\leadsto} v$ which is shown in Fig. 6(a). Therefore, G is a simple cycle.

Case 2: $e \in \{e''_i, e''_j\}$; that is, e''_i or e''_j is the bridge. Without loss of generality, we assume that e''_i is the bridge. Since G_1 and G_2 are simple cycles, the paths $u_p \overset{G'_1}{\leadsto} v_q$ and $u_{p'} \overset{G'_2}{\leadsto} v_{q'}$ exist. Hence, after removing e''_i, there is still a path that is $u_p \overset{G'_1}{\leadsto} v_q \leftrightarrow v_{q'} \overset{G'_2}{\leadsto} u_{p'}$ which is shown in Fig. 6(b). Therefore, G is a simple cycle.

Based on the above argument, G is connected and does not contain a bridge. G is a simple cycle, this is a contradiction. Hence, the result holds. □

Algorithm 1 circular-swap$(e_0, \ldots, e_{h-1})$

```
1: Input: $h$ legal edges.
2: Output: A set of edges derived by the circular-swap function.
3: **procedure** circular-swap$(e_0, \ldots, e_{h-1})$
4: $E' = \emptyset, E^1_{out} = \emptyset, E^2_{out} = \emptyset$;
5: **if** $h = 2$ **then**
6:     $E' = \text{swap}(e_0, e_1)$;
7: **end if**
8: **if** $h = 3$ **then**
9:     Let $\{e'_0, e'_1\} = \text{swap}(e_0, e_1)$;
10:    $E' = (\text{swap}(e_0, e_1) - \{e'_0\}) \cup \text{swap}(e'_0, e_2) = \{e'_1\} \cup \text{swap}(e'_0, e_2)$;
11: **end if**
12: **if** $h > 3$ **then**
13:    $E^1_{out}$ =circular-swap$(e_0, \ldots, e_{\lfloor (h-1)/2 \rfloor})$;
14:    $E^2_{out}$ =circular-swap$(e_{\lfloor (h-1)/2 \rfloor + 1}, \ldots, e_{h-1})$;
15:    Let $e''_0 = e'_i$, where $e'_i$ is an arbitrary edge selected from $E^1_{out}$;
16:    Let $e''_1 = e'_j$, where $e'_j$ is an arbitrary edge selected from $E^2_{out}$;
17:    $E' = (E^1_{out} \cup E^2_{out} - \{e''_0, e''_1\}) \cup \text{swap}(e''_0, e''_1)$;
18: **end if**
19: **return** $E'$;
20: **end procedure**
```

Lemma 4. circular-swap$(e_0, \ldots, e_{h-1}) = \{e'_0, \ldots, e'_{h-1}\}$ *can be found in sequential linear time or in* $O(\log h)$ *parallel time.*

Proof. We apply Algorithm 1 to compute circular-swap$(e_0, \ldots, e_{h-1})$. First, we prove the algorithm's correctness. Let E' be the set of edges returned by the algorithm, where each edge in E' is legal. We prove that E' satisfies the properties defined in Definition 2 by analyzing the following cases.

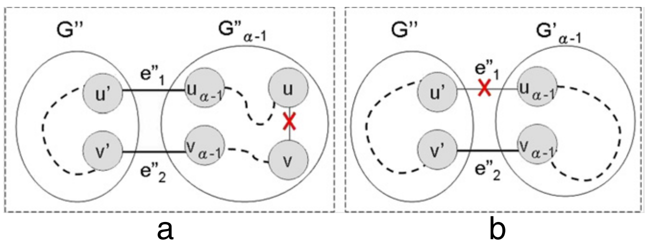

Fig. 7. (a) e is in $G''_{\alpha-1}$. (b) $e = e''_1 = (u', u_{\alpha-1})$.

Case 1: When $h = 2$, $E' = \text{swap}(e_0, e_1)$. Lemma 1 shows that $\text{swap}(e_0, e_1)$ exists and that E' has the needed properties.

Case 2: When $h = 3$, $E' = \text{circular-swap}(e_0, e_1, e_2)$. Lemma 2 shows that $\text{circular-swap}(e_0, e_1, e_2)$ exists and that E' has the needed properties.

Case 3: When $h > 3$, $E' = \text{circular-swap}(e_0, \ldots, e_{h-1})$. $\text{circular-swap}(e_0, \ldots, e_{h-1})$ exists by Lemma 3 and that E' has the needed properties.

Next, we prove that Algorithm 1 runs in sequential linear time or in $O(\log h)$ parallel time. According to the algorithm, a set of the legal edges are swapped in parallel, and the size of the set is reduced by half each time a recursive call is made. It is obvious that both $\text{circular-swap}(e_0, e_1)$ and $\text{circular-swap}(e_0, e_1, e_2)$ run in constant time. There are at most $O(\log h)$ recursive calls, and the i-th recursive call performs 2^i swap operations, where $0 \le i \le \log h$; that is, $O(2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{\log h}) = O(h)$. Hence, it takes a total of $O(h)$ time for all operations. Note that each recursive call is made in constant time on an EREW PRAM platform. Therefore, $\text{circular-swap}(e_0, \ldots, e_{h-1}) = \{e'_0, \ldots, e'_{h-1}\}$ can be found in sequential linear time or in $O(\log h)$ parallel time. □

Theorem 2. *Given 2-edge-connected components $G_0, \ldots, G_{h-1}$, and a legal edge $e_i = (u_i, v_i) \in G_i$, let $G = (G_0 \cup \cdots \cup G_{h-1} - \{e_0, \ldots, e_{h-1}\}) \cup \text{circular-swap}(e_0, \ldots, e_{h-1})$, where $h \ge 2$ and $0 \le i \le h-1$. Then, G must be a 2-edge-connected component.*

Proof. Assume that this theorem is not correct; that is, there exists a bridge $e = (u, v)$ in G. After removing e, there will not be a path from u to v. We prove the theorem by induction on h, where $h \ge 2$.
Inductive basis: According to Theorem 1, G is a 2-edge-connected component when $h = 2$.
Inductive step: Let $e_i = (u_i, v_i)$ be a legal edge, and let $G'_i = G_i - \{e_i\}$, where $0 \le i \le h-1$. We assume that $G' = (G_0 \cup \cdots \cup G_{\alpha-2} - \{e_0, \ldots, e_{\alpha-2}\}) \cup \text{circular-swap}(e_0, \ldots, e_{\alpha-2})$ is a 2-edge-connected component for $h = \alpha - 1$, where α is a positive integer larger than 2.

For $h = \alpha$, we prove that $G = (G_0 \cup \cdots \cup G_{\alpha-1} - \{e_0, \ldots, e_{\alpha-1}\}) \cup \text{circular-swap}(e_0, \ldots, e_{\alpha-1})$ is a 2-edge-connected component. Let $e'_i \in \text{circular-swap}(e_0, \ldots, e_{\alpha-2})$. Without loss of generality, we assume that $e'_i = e'_{\alpha-2}$. In addition, we let $\text{circular-swap}(e_0, \ldots, e_{\alpha-1}) \longleftarrow (\text{circular-swap}(e_0, \ldots, e_{\alpha-2}) - \{e'_{\alpha-2}\}) \cup \text{swap}(e'_{\alpha-2}, e_{\alpha-1})$ if $\alpha > 2$ by Lemma 2. Hence, $G = (G' \cup G_{\alpha-1} - \{e'_{\alpha-2}, e_{\alpha-1}\}) \cup \text{swap}(e'_{\alpha-2}, e_{\alpha-1})$ for $h = \alpha$, where $\alpha > 2$. Let $\text{swap}(e'_{\alpha-2}, e_{\alpha-1}) = \{e''_1, e''_2\}$ and $\text{END}(e'_{\alpha-2}) = (u', v')$ and $G'' = G' - \{e'_{\alpha-2}\}$. Without loss of generality, we assume that $e''_1 = (u', u_{\alpha-1})$ and $e''_2 = (v', v_{\alpha-1})$. We consider the following two cases.

Case 1. $e \in G'_i$. Assuming that G' is a 2-edge-connected component, then, according to Theorem 1, there is no bridge e in G'_i or G_i, where $0 \le i \le \alpha - 2$. Without loss of generality, we assume that $e \in G'_{\alpha-1}$. Since G' is a 2-edge-connected component and $G'_i = G_i - \{e_i\}$, where $0 \le i \le \alpha - 1$, there exist paths $u \overset{G'_{\alpha-1}}{\rightsquigarrow} u_{\alpha-1}$, $v \overset{G'_{\alpha-1}}{\rightsquigarrow} v_{\alpha-1}$, and $u' \overset{G''}{\rightsquigarrow} v'$. Let $G''_{\alpha-1} = G'_{\alpha-1} - \{e\}$. Hence, after removing e, a path still exists from u to v through $u \overset{G''_{\alpha-1}}{\rightsquigarrow} u_{\alpha-1} \leftrightarrow u' \overset{G''}{\rightsquigarrow} v' \leftrightarrow v_{\alpha-1} \overset{G''_{\alpha-1}}{\rightsquigarrow} v$ which is shown in Fig. 7(a). This means G is connected and e is not a bridge in G'_i.

Case 2. $e \in \{e''_1, e''_2\}$. Without loss of generality, we assume that $e = e''_1$ and let $\{u, v\} = \{u', u_{\alpha-1}\}$. Note that G' and $G_{\alpha-1}$ are connected. In other words, there exist paths $u' \overset{G''}{\rightsquigarrow} v'$ and $u_{\alpha-1} \overset{G'_{\alpha-1}}{\rightsquigarrow} v_{\alpha-1}$. After removing e, there is still a path from u to v through $u' \overset{G''}{\rightsquigarrow} v' \leftrightarrow v_{\alpha-1} \overset{G'_{\alpha-1}}{\rightsquigarrow} u_{\alpha-1}$ which is shown in Fig. 7(b). This means G' is connected, and e''_1 and e''_2 are not bridges. Hence, the theorem holds. □

4. Main result

Let G be an input graph in which a *loose block*, B, is a 2-edge-connected component so that the degree of the corresponding vertex in $\text{BB}(G)$ is at most 1. In $\text{BB}(G)$, a *loose vertex*, b, is the vertex that corresponds to the loose block B in G. Hereafter, we use F and $\text{BB}(G)$ interchangeably to denote the bridge-block forest of an input graph G. Recall that the vertices in G are partitioned into $P_1, P_2, \ldots, P_k$. Let S_i and H denote the sets of mono leaves of partition P_i and the set of hybrid leaves in $\text{BB}(G)$ respectively; and let S_i^* and H^* denote the sets of mono isolated vertices of P_i and the set of hybrid isolated vertices in $\text{BB}(G)$ respectively. In addition, let $s_{max} = \max\{|S_i| + 2|S_i^*| \,|\, 1 \le i \le k\}$. Without loss of generality, we assume that $s_{max} = |S_1| + 2|S_1^*|$. Then, we say that $\text{BB}(G)$ is *P_i-dominated* if there exists an i such that $|S_i| + 2|S_i^*| > \lceil (2|S_1^*| + \cdots + 2|S_k^*| + 2|H^*| + |S_1| + \cdots + |S_k| + |H|)/2 \rceil$.

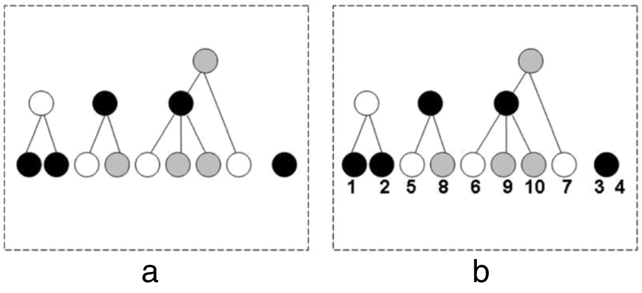

Fig. 8. The illustration of the numbering procedure:(a) the given graph; (b) the result of the numbering procedure.

4.1. *Lower bound on* $|\text{aug2}e(\text{BB}(G))|$

Let $\text{LOW}(\text{BB}(G)) = \max\{s_{max}, \lceil(2|S_1^*| + \cdots + 2|S_k^*| + 2|H^*| + |S_1| + \cdots + |S_k| + |H|)/2\rceil\}$.

Theorem 3. $|\text{aug2}e(\text{BB}(G))| \geq \text{LOW}(\text{BB}(G))$.

Proof. To make the resulting graph 2-edge-connected, each leaf must have at least one incident edge and each isolated vertex must have at least two incident edges. That is, $|\text{aug2}e(\text{BB}(G))| \geq \lceil(2|S_1^*| + \cdots + 2|S_k^*| + 2|H^*| + |S_1| + \cdots + |S_k| + |H|)/2\rceil$. By Fact 2, two endpoints of an added edge cannot be in the same vertex partition. Thus, $|\text{aug2}e(\text{BB}(G))| \geq s_{max}$, which means the theorem holds. □

Corollary 1. *If* BB(G) *is* P_i*-dominated, then* $\text{LOW}(\text{BB}(G)) = |S_i| + 2|S_i^*|$.

Proof. We can derive the corollary by the definitions of P_i-dominated and LOW(BB(G)). □

4.2. *Augmentation algorithm*

The steps of the algorithm are as follows. First, we add LOW(BB(G)) legal edges to the graph BB(G) according to the indexes assigned to the leaves and isolated vertices of an input bridge-block forest, as shown in Algorithm 2. Then, based on the assigned numbers, we add edges between the leaves and isolated vertices of the input graph without violating the partition constraint.

The resulting graph, G', consists of h 2-edge-connected components. If $h = 1$, then we terminate the process; otherwise, we arbitrarily select one legal edge for each loose block in G', and perform a circular-swap operation on the chosen edges. By replacing the original set of the chosen edges with the set of edges obtained by the circular swap operation, the resulting graph is guaranteed to be 2-edge-connected.

Here, we assume that BB(G) contains at least two leaves in different vertex partitions, or two isolated vertices in different vertex partitions, or one leaf and one isolated vertex in different vertex partitions.

Algorithm 2 Numbering the leaves and isolated vertices in F

```
1:  Input: $F$ is a bridge-block forest with $k$ partitions of its vertices.
2:  Output: The enumerated leaves and isolated vertices of $F$.
3:  procedure Numbering($F$)
4:  Let $\lambda_0 = 0$;
5:  for $i$ from 1 to $k$ do
6:      Assign a number to each leaf in $S_i$ from $\lambda_{i-1} + 1$ to $\lambda_{i-1} + |S_i|$;
7:      Assign two consecutive numbers to each isolated vertex in $S_i^*$ from $\lambda_{i-1} + |S_i| + 1$ to $\lambda_{i-1} + |S_i| + 2|S_i^*|$;
8:      Let $\lambda_i = \lambda_{i-1} + |S_i| + 2|S_i^*|$;
9:  end for
10: Assign a number to each leaf in $H$ from $\lambda_k + 1$ to $\lambda_k + |H|$;
11: Assign two consecutive numbers to each isolated vertex in $H^*$, from $\lambda_k + |H| + 1$ to $\lambda_k + |H| + 2|H^*|$;
12: end procedure
```

Fig. 8 illustrates the numbering procedure in Algorithm 2. In this example, there are 9 vertices, one of which is an isolated vertex. The 9 vertices are divided into three vertex partitions. The vertices with black, white, and gray colors denote the vertices of the first, second, and third partitions respectively. We assign the numbers 1 and 2 to the two black vertices; and the black isolated vertex is assigned the numbers 3 and 4. We also assign the numbers 5, 6 and 7 to the three white vertices, and the numbers 8, 9 and 10 to the three gray vertices in Fig. 8(b).

Lemma 5. *Given a bridge-block forest* BB(G) *with k partitions of its vertices, Algorithm 2 maintains the following properties.*

1. *For each vertex partition* P_i*,* $1 \leq i \leq k$*, vertices in the same vertex partition are assigned consecutive numbers.*

2. *All vertices are assigned distinct numbers.*
3. *In total, $|S_i| + 2|S_i^*|$ numbers are assigned to degree ≤ 1 vertices in partition P_i, where $1 \leq i \leq k$.*

Proof. The proof is straightforward based on Algorithm 2. □

Corollary 2. *Let $\ell = \sum_{i=1}^{k} |S_i| + 2|S_i^*| + |H| + 2|H^*|$ denote all the numbers assigned to degree ≤ 1 vertices in* BB(G). *The following properties can be inferred from Lemma 5.*

1. *When ℓ is even, if the vertices in any partition are assigned no more than $\ell/2$ numbers, then two vertices with numbers j and $j + \ell/2$ must be in different vertex partitions for any j, s.t., $1 \leq j \leq \ell/2$.*
2. *When ℓ is odd, if the vertices in any partition are assigned no more than $\lfloor \ell/2 \rfloor$ numbers, then two vertices with numbers j and $j + \lfloor \ell/2 \rfloor$ must be in different vertex partitions for any j, s.t., $1 \leq j \leq \lfloor \ell/2 \rfloor$.*

Proof. The proof is straightforward based on Lemma 5. □

Next, we propose our main algorithm, Algorithm 3, for finding a smallest 2-edge-connectivity augmentation of a graph G with a partition constraint.

Algorithm 3 Finding a smallest 2-edge-connectivity augmentation of an input graph G with a partition constraint.

```
Input: A graph $G$.
Output: A smallest 2-edge-connectivity augmentation of $G$ with a partition constraint.
procedure FS2Aug($G$)
Let $F = \text{BB}(G)$;
Let $\ell = \sum_{i=1}^{k} |S_i| + 2|S_i^*| + |H| + 2|H^*|$;
$E = \emptyset$; $E' = \emptyset$; $E_1 = \emptyset$; $E_2 = \emptyset$;
Numbering($F$);
Let $v_i$ denote a vertex with the number $i$;
switch($s_{max}$)
Case 1: $s_{max} \leq \lfloor \ell/2 \rfloor$
  Case 1.1: $\ell$ is even
    $E' = \{(v_i, v_{i+\ell/2}) | 1 \leq i \leq \ell/2\}$;
  Case 1.2: $\ell$ is odd
    $E' = \{(v_i, v_{i+\lfloor \ell/2 \rfloor}) | 1 \leq i \leq \lfloor \ell/2 \rfloor\}$;
    $E_1 = \{v_{\lfloor \ell/2 \rfloor}, v_\ell\}$;
Case 2: $s_{max} > \lfloor \ell/2 \rfloor$
  $E' = \{(v_i, v_{i+s_{max}}) | 1 \leq i \leq \ell - s_{max}\}$;
  $E_1 = \{(v_j, v_\ell) | \ell - s_{max} + 1 \leq j \leq s_{max}\}$;
Let $E' = E' \cup E_1$;
Let $F' = \text{BB}(F \cup E')$;
Let $X$ be the set of loose vertices in $F'$;
if $|X| > 1$ then
    For each loose vertex in $F'$, arbitrarily select an added edge from the corresponding loose block in $F \cup E'$; let $E_2$ denote
        the set of selected added edges, here $E_2 = \{e_0, \ldots, e_{|X|-1}\}$;
    $E = (E' - E_2) \cup$ circular-swap($e_0, e_1, \ldots, e_{|X|-1}$);
else
    $E = E'$;
end if
return $E$;
end procedure
```

Lemma 6. *Each added edge in E' found in step 19 of Algorithm 3 is legal.*

Proof. By Lemma 5, vertices in the same vertex partition are assigned consecutive numbers, and the sequence of numbers assigned to degree ≤ 1 vertices in P_i partition is $|S_i| + 2|S_i^*|$, where $1 \leq i \leq k$. Note that $s_{max} = |S_1| + 2|S_1^*|$. We prove the lemma by the following two cases.

Case 1: $s_{max} \leq \lfloor \ell/2 \rfloor$. Since $s_{max} = \max\{|S_i| + 2|S_i^*| | 1 \leq i \leq k\}$, by Corollary 2, no vertex partition is assigned more than $\ell/2$ numbers if ℓ is even; and no vertex partition is assigned more than $\lfloor \ell/2 \rfloor$ numbers if ℓ is odd. Next, we analyze these two subcases.

Case 1.1: ℓ is even. The algorithm only adds an edge between two vertices with numbers j and $j + \ell/2$ for any j, s.t., $1 \leq j \leq \ell/2$. Since no vertex partition is assigned more than $\ell/2$ consecutive numbers, by Corollary 2-1, two vertices with numbers j and $j + \ell/2$ must be in different vertex partitions for any j, s.t., $1 \leq j \leq \ell/2$. Therefore, the added edges are legal.

Case 1.2: ℓ is odd. The algorithm adds an edge between two vertices with numbers j and $j + \lfloor \ell/2 \rfloor$ for any j, s.t., $1 \leq j \leq \lfloor \ell/2 \rfloor$, and between two vertices with numbers $\lfloor \ell/2 \rfloor$ and ℓ. Since no vertex partition is assigned more than $\lfloor \ell/2 \rfloor$

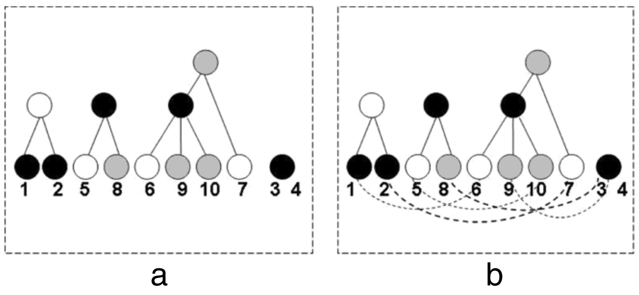

Fig. 9. (a) The result of the Numbering Procedure in the corresponding step 7 of Algorithm 3. (b) The corresponding steps 9–18 of Algorithm 3.

consecutive numbers, by Corollary 2-2, two vertices with numbers j and $j + \lfloor \ell/2 \rfloor$ must be in different vertex partitions for any j, s.t., $1 \le j \le \lfloor \ell/2 \rfloor$. Therefore, the added edges are legal.

Case 2: $s_{max} > \lfloor \ell/2 \rfloor$. In this case, our algorithm only adds edges between a vertex in the vertex partition that has s_{max} numbers and another vertex in a different vertex partition. Therefore, the lemma holds. □

Fig. 9 illustrates steps 7–18 of Algorithm 3. To refer to the example in Fig. 8, Fig. 9(a) shows the result of the Numbering Procedure called in step 7 of Algorithm 3. Then, we add edges based on the following vertex pairs: $(1, 6)$, $(2, 7)$, $(3, 8)$, $(4, 9)$ and $(5, 10)$ in Fig. 9(b). In this example, it is clear that the partition constraint holds for the five added edges.

Theorem 4. *Given an input graph G, Algorithm 3 adds* LOW(BB(G)) *edges.*

Proof. We prove that the number of edges added in Algorithm 3 is minimized by analyzing the following two cases. Note that $\ell = \sum_{i=1}^{k} |S_i| + 2|S_i^*| + |H| + 2|H^*|$ and LOW(BB(G))= $\max\{s_{max}, \lceil \ell/2 \rceil\}$.

Case 1: $s_{max} \le \lfloor \ell/2 \rfloor$; Clearly, the number of added edges is equal to $\lceil \ell/2 \rceil$ by steps 10–15 of Algorithm 3. Therefore, the number of added edges in this case is equal to LOW(BB(G)).

Case 2: $s_{max} > \lfloor \ell/2 \rfloor$. Here, the number of added edges is equal to s_{max}. Hence, the number of added edges in this case is also equal to LOW(BB(G)).

Note that no edge is added in steps 19–26. Therefore, by Theorem 3, Algorithm 3 adds LOW(BB(G)) edges. □

Let F be a bridge-block forest of an input graph G, and let E' be a set of added edges computed in step 19 of Algorithm 3. We prove that the resulting graph $F \cup E'$ computed by the algorithm is 2-edge-connected or that each block corresponding to a loose vertex in BB($F \cup E'$) contains a newly added edge by Lemma 7. Then, we apply the circular-swap function introduced in Section 3 to obtain a 2-edge-connected graph by Lemma 8.

Lemma 7. *Let F be a bridge-block forest of an input graph G. Every block corresponding to a loose vertex in* BB($F \cup E'$) *contains an edge in E', where E' is a set of added edges computed in step* 19 *of Algorithm* 3*.*

Proof. We assume that this theorem is incorrect; that is, there exists a loose vertex in BB($F \cup E'$) whose corresponding block B is in $F \cup E'$ does not contain an added edge. Note that the block B with respect to a loose vertex in BB($F \cup E'$) is either the case that B is a block of F, i.e., $B \in F$, or the case that B is not a block of F, i.e., $B \notin F$, but B contains some blocks in F. We prove this lemma by analyzing the two following cases.

Case 1: $B \in F$. Since the block B does not contain an edge in E' and the vertex corresponding to B in BB($F \cup E'$) is a loose vertex, the degree of B in F is at most 1. Hence, B is also a loose vertex in F and the block in G corresponding to B is a loose block. Here, we prove that B contains an edge in E'. Note that in Algorithm 2, we assign numbers to all the loose blocks in G. Let v_i denote the loose vertex with the number i in F. Edges are added to all loose vertices, as shown in steps $9 - 18$ of Algorithm 3. By Lemma 6, an added edge is legal and connects v_i and v_j, where v_j is assigned according to Cases 1 and 2 discussed in Algorithm 3. Therefore, a loose vertex B is connected to another vertex by an added edge. However, this contradicts the assumption that B is in $F \cup E'$ and does not contain any edges in E'.

Case 2: $B \notin F$, but B contains blocks in F. Since B contains a set of the blocks in F, it is clear that those blocks can be merged by using newly added edges in E'. Therefore, B contains newly added edges in E'.

According to above analyses, each loose block corresponding to a loose vertex in BB($F \cup E'$) contains an edge in E', which is the set of added edges computed in step 19 of Algorithm 3. □

Lemma 8. *Let G be an input graph; and let $F =$* BB(G) *and $F' =$* BB($F \cup E'$)*, where E' is the set of added edges computed in step* 19 *of Algorithm* 3*. $F' \cup E$ is* 2*-edge-connected without violating the partition constraint, where E is the set of added edges returned by the algorithm.*

Proof. By Lemmas 2 and 6, all added edges are legal. We consider two cases to determine whether the resulting graph $F' \cup E$ is 2-edge-connected.

Case 1: Assume that there exists a bridge $e = (u, v)$, where $e \in E$ and u and v are leaves in F'. By Theorem 2, all leaves in F' are connected by a cycle C in steps 20–26 of Algorithm 3. Since there exists another path from u to v that passes through C, no added edge is a bridge.

Case 2: Assume that any branch of F' is a bridge $e = (u, v)$. Since all leaves in F' are connected by E, there exists another path from u to v that passes through C. Hence, no branch of F' is a bridge.

Therefore, without violating the partition constraint, $F' \cup E$ is 2-edge-connected. □

Theorem 5. *Algorithm 3 runs in sequential linear time or in $O(\log n)$ parallel time on an EREW PRAM using a linear number of processors.*

Proof. Given a graph G as an input, by Fact 1, the first step in Algorithm 3 takes sequential linear time or takes $O(\log n + T_M(n, m))$ parallel time to compute $\mathrm{BB}(G)$ using $O((n+m)/\log n + P_M(n, m))$ processors on an EREW PRAM. After computing $\mathrm{BB}(G)$, the numbering procedure takes sequential linear time or $O(\log n)$ parallel time. Then, the algorithm takes $O(1)$ time to determine which case should be executed. In steps 7–18, the algorithm takes sequential linear time or $O(\log n)$ parallel time to add edges between vertices. Finally, it applies the concept of Algorithm 1 to reconnect the graph to form a 2-edge-connected component. By Lemma 4, Algorithm 1 takes sequential linear time or $O(\log h)$ parallel time, where $h \le O(n)$. Therefore, the theorem holds. □

Lemma 9. *Let G be a k-partite graph but not a 2-edge-connected graph, $G \cup \mathrm{F_n}(\mathrm{aug2}e(\mathrm{BB}(G)))$ is a simple graph if any one of following conditions is satisfied:*

Case 1: $\mathrm{BB}(G)$ contains at least three vertices and there are three vertex partitions in $\mathrm{BB}(G)$, each of which contains at least one vertex.

Case 2: $\mathrm{BB}(G)$ contains at least four vertices and there are two vertex partitions in $\mathrm{BB}(G)$, each of which contains at least two vertices.

Case 3: $\mathrm{BB}(G)$ contains at least one hybrid vertex.

Proof. We prove this lemma case by case as follows.

Case 1. Without loss of generality, let u, v and w denote vertices of $\mathrm{BB}(G)$ in P_1, P_2 and P_3, respectively. For an isolated vertex x in $\mathrm{BB}(G)$, without loss of generality, we assume $x \in P_1$, then an augmentation may contain edges (v, x) and (w, x). For a leaf y in $\mathrm{BB}(G)$, an augmentation may contain an edge (y, z), where $z \in \{u, v, w\}$, z and y are in different vertex partitions, and z is not the parent of y. Therefore, $G \cup \mathrm{F_n}(\mathrm{aug2}e(\mathrm{BB}(G)))$ is a simple graph.

Case 2. Without loss of generality, let u_1 and u_2 denote two vertices in P_1, and v_1 and v_2 denote two vertices in P_2. For an isolated vertex x in $\mathrm{BB}(G)$, without loss of generality, we assume $x \in P_1$, then an augmentation may contain edges (v_1, x) and (v_2, x). For a leaf y in $\mathrm{BB}(G)$, an augmentation may contain an edge (y, z), where $z \in \{u_1, u_2, v_1, v_2\}$, z and y are in different vertex partitions, and z is not the parent of y. Therefore, $G \cup \mathrm{F_n}(\mathrm{aug2}e(\mathrm{BB}(G)))$ is a simple graph.

Case 3. If $\mathrm{BB}(G)$ contains a hybrid vertex, then any one of following subcases is true. Note that a hybrid vertex in $\mathrm{BB}(G)$ is a block in G. Let this hybrid vertex be Q.

Case 3.1: Q contains at least three vertices and at least three vertex partitions, and there are three vertices in three different vertex partitions.

For Case 3.1, let u, v, and w denote three vertices in the block of G which corresponds to Q. Without loss of generality, we assume $u \in P_1$, $v \in P_2$, and $w \in P_3$. For an isolated vertex x in $\mathrm{BB}(G)$, without loss of generality, we assume $x \in P_1$, then an augmentation may contain edges between Q and x in $\mathrm{BB}(G)$. So that, the corresponding edges found by F_n are (v, x') and (w, x'), where x' is a vertex in G which corresponds to x, $v \in P_2$ and $w \in P_3$. For a leaf y in $\mathrm{BB}(G)$, an augmentation may contain an edge (y, Q) in $\mathrm{BB}(G)$, so that the corresponding edge found by F_n is (y', z), where y' is a vertex in G which corresponds to y and $z \in \{u, v, w\}$, z and y' are in different vertex partitions, and z is not the parent of y'. Therefore, $G \cup \mathrm{F_n}(\mathrm{aug2}e(\mathrm{BB}(G)))$ is a simple graph.

Case 3.2: Q contains at least four vertices and at least two vertex partitions, and there are two different vertex partitions, each of which contains at least two vertices.

For Case 3.2, let u_1, u_2, v_1, and v_2 denote four vertices in the block of G which corresponds to Q, where $u_1, u_2, \in P_1$, $v_1, v_2 \in P_2$.

For an isolated vertex x in $\mathrm{BB}(G)$, without loss of generality, assume $x \in P_1$, then an augmentation may contain edges between Q and x in $\mathrm{BB}(G)$. So that, the corresponding edges found by F_n are (v_1, x') and (v_2, x'), where x' is a vertex in G corresponding to x and $v_1, v_2 \in P_2$. For a leaf y in G, an augmentation may contain an edge (y, Q) in $\mathrm{BB}(G)$, so that, the corresponding edge found by F_n is (y', z), where y' is a vertex in G corresponding to y and $z \in \{u_1, u_2, v_1, v_2\}$, z and y' are in different vertex partitions, and z is not the parent of y'. Therefore, $G \cup \mathrm{F_n}(\mathrm{aug2}e(\mathrm{BB}(G)))$ is a simple graph.

Hence, the lemma holds. □

4.3. Other cases

Let G be a k-partite but not 2-edge-connected graph, we consider two cases of $\mathrm{BB}(G)$ not discussed in Lemma 9.

Case 1: $\mathrm{BB}(G)$ contains two vertex partitions, but none of its vertices is a hybrid vertex. Without loss of generality, we assume that the two vertex partitions of $\mathrm{BB}(G)$ are P_1 and P_2, and $|P_1| \ge |P_2|$. This case can be divided into three subcases by considering the vertices in $\mathrm{BB}(G)$.

Case 1.1: $\mathrm{BB}(G)$ consists of exactly two vertices. Let u, v denote two vertices of $\mathrm{BB}(G)$, then an augmentation would add edges between u and v. Therefore, the resulting graph must be a multigraph.

Case 1.2: $BB(G)$ consists of exactly three vertices. Let u_1, u_2, v denote the three vertices of $BB(G)$. Without loss of generality, assume that u_1, u_2 are in P_1 and v is in P_2. Then, an augmentation would add edges between u_1 and v, and add edges between u_2 and v. Therefore, the resulting graph must be a multigraph.

Case 1.3: $BB(G)$ contains at least four vertices and $|P_2| = 1$. In this subcase, let u denote the only vertex of $BB(G)$ that is in P_2. Then, an augmentation would add edges between u and all other vertices of $BB(G)$. Therefore, the resulting graph must be a multigraph.

Case 2: Finally, we consider a trivial case where the input graph $BB(G)$ contains only one partition of its vertices, i.e., all vertices are mono vertices. In this case, unless the input graph is already 2-edge-connected, there is no 2-edge-connectivity augmentation solution.

The analyses in Lemma 9 and Section 4.3 demonstrate the properties of a given graph $BB(G)$.

5. Concluding remarks

The main algorithm proposed in this paper produces a simple graph if such an augmentation solution exists. If this is not possible, it generates a multigraph. First, we prove that we can easily add edges to the input graph such that the resulting graph G' has the property whereby each degree ≤ 1 block of $BB(G')$ contains a legal edge. We select a legal edge of each degree ≤ 1 block and apply the circular-swap function to all the selected legal edges to form a 2-edge-connected graph. The main algorithm can be parallelized to run in $O(\log n)$ parallel time using a linear number of EREW processors.

Acknowledgements

We thank anonymous referees for very helpful comments.

The second, third, and fifth authors were supported in part by National Science Council (Taiwan) Grants NSC 97-2221-E-001-011-MY3.

References

[1] A. Frank, Connectivity augmentation problems in network design, in: J. Birge, K. Murty (Eds.), Mathematical Programming: State of the Art 1994, The University of Michigan, 1994, pp. 34–63.
[2] T.-s. Hsu, Graph augmentation and related problems: theory and practice, Ph.D. thesis, University of Texas at Austin, TX, USA, 1993.
[3] H. Nagamochi, Recent development of graph connectivity augmentation algorithms, IEICE Transactions on Information and Systems 83 (3) (2000) 372–383.
[4] T.-s. Hsu, M.-Y. Kao, Optimal augmentation for bipartite componentwise biconnectivity in linear time, SIAM Journal on Discrete Mathematics 19 (2) (2005) 345–362.
[5] J. Bang-Jensen, H.N. Gabow, T. Jordán, Edge-connectivity augmentation with partition constraints, SIAM Journal on Discrete Mathematics 12 (2) (1999) 160–207.
[6] M.-Y. Kao, Linear-time optimal augmentation for componentwise bipartite-completeness of graphs, Information Processing Letters 54 (1) (1995) 59–63.
[7] N.R. Adam, J.C. Worthmann, Security-control methods for statistical databases: a comparative study, ACM Computing Surveys 21 (4) (1989) 515–556.
[8] F.Y. Chin, G. Ozsoyoglu, Auditing and inference control in statistical databases, IEEE Transactions Software Engineering 8 (6) (1982) 574–582.
[9] L.H. Cox, Suppression methodology and statistical disclosure control, Journal of the American Statistical Association 75 (370) (1980) 377–385.
[10] D.E. Denning, J. Schlörer, Inference controls for statistical databases, IEEE Computer 16 (1983) 69–82.
[11] B.L.G.J.P. Kelly, A.A. Assad, Cell suppression: disclosure protection for sensitive tabular data, Networks 22 (4) (1992) 397–417.
[12] D. Gusfield, A graph theoretic approach to statistical data security, SIAM Journal on Computing 17 (3) (1988) 552–571.
[13] M.-Y. Kao, Data security equals graph connectivity, SIAM Journal on Discrete Mathematics 9 (1) (1996) 87–100.
[14] M.-Y. Kao, Total protection of analytic-invariant information in cross-tabulated tables, SIAM Journal on Computing 26 (1) (1997) 231–242.
[15] K. Eswaran, R. Tarjan, Augmentation problems, SIAM Journal on Computing 5 (1976) 653–665.
[16] T.-s. Hsu, Simpler and faster biconnectivity augmentation, Journal of Algorithms 45 (1) (2002) 55–71.
[17] A. Rosenthal, A. Goldner, Smallest augmentations to biconnect a graph, SIAM Journal on Computing 6 (1977) 55–66.
[18] T. Watanabe, A. Nakamura, A minimum 3-connectivity augmentation of a graph, Journal of Computer and System Science 46 (1) (1993) 91–128.
[19] T.-s. Hsu, On four-connecting a triconnected graph, Journal of Algorithm 35 2 (2000) 202–234.
[20] P.-C. Huang, H.-W. Wei, W.-C. Lu, W.-K. Shih, T.-s. Hsu, Smallest bipartite bridge-connectivity augmentation, Algorithmica 54 (3) (2009) 353–378.
[21] F. Harary, Graph Theory, in: Addison-Wesley Series in Mathematics, Reading, 1969.
[22] K. Chong, Y. Han, T. Lam, Concurrent threads and optimal parallel minimum spanning trees algorithm, Journal of the ACM 48 (2) (2001) 297–323.
[23] R. Cole, U. Vishkin, Approximate parallel scheduling. Part II: Applications to logarithmic-time optimal graph algorithms, Information and Computation 92 (1) (1991) 1–47.
[24] V. Ramachandran, Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity, in: J. Reif (Ed.), Synthesis of Parallel Algorithms, Morgan-Kaufmann, 1993, pp. 275–340.
[25] R.E. Tarjan, Depth-first search and linear graph algorithms, SIAM Journal on Computing 1 (1972) 146–160.
[26] R. Tarjan, U. Vishkin, An efficient parallel biconnectivity algorithm, SIAM Journal on Computing 14 (4) (1985) 862–874.