Bridge-connectivity Augmenting Problem with a Partition Constraint

Hsin-Wen Wei*, Wan-Chen Lu†, Pei-Chi Huang†, Wei-Kuan Shih†, and Tsan-sheng Hsu*
*Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan
{hwwei, tshsu}@iis.sinica.edu.tw
†Department of Computer Science, National Tsing-Hua University, Hsinchu, Taiwan
{wanchen, peggy, wshih}@rtlab.cs.nthu.edu.tw

Abstract

This paper considers 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 partition constraint. To solve this problem, we propose a simple linear-time algorithm. We show that the algorithm runs in $O(\log n)$ parallel time on an EREW PRAM using a linear number of processors.

Key words: 2-edge-connectivity, bridge-connectivity, augmentation, partition constraint

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 [4, 7, 12] for a comprehensive survey. 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 [3, 8, 9, 10, 14, 17]. Note that there is a linear-time algorithm for the smallest bridge-connectivity augmentation problem on the general graph that does not have a partition constraint [3]. In [6], a linear-time algorithm for bridge-connectivity augmentation with a bipartite constraint is described. In [11], Jensen et al. proposed a polynomial time algorithm that solves the k-edge-connectivity augmentation problem on a graph that has partition constraints in $O(n(m+n\log n)\log n)$ time, where m is the number of distinct edges in the input graph.

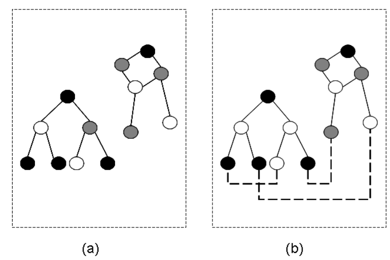

Figure 1: (a) A graph with three partitions of vertices. (b) A smallest 2-edge-connectivity augmentation of (a) with the set of added edges marked by dashed lines.

In this paper, we focus on augmenting graphs with a partition constraint. Here the partition constraint is that the vertex set of an input graph is 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 given graph with a given partition constraint to make it 2-edge-connected, or bridge-connected, while maintaining the constraint. Figure 1(a) shows an example of a graph with three partitions of vertices. A smallest 2-edge-connectivity augmentation of Figure 1(a) is shown in Figure 1(b).

We solve the problem of 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* [5]. Our approach adds the smallest possible number of edges to make a bridge-block forest 2-edge-connected. Note that 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 algorithm runs in sequential liner time and $O(\log n)$ parallel time on an EREW PRAM using a linear number of processors.

The remainder of this paper is organized as follows. Section 2 contains graph-theoretical definitions and previously known properties. In Section 3, we introduce the concept of loose edges and propose an algorithm that makes a loose forest 2-edge-connected. In Section 4, we propose an algorithm that finds a smallest 2-edge-connectivity augmentation for a bridge-block forest. The paper is concluded in Section 5.

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, connected, and acyclic graph. 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. A partition of the vertex set in a graph is called a *vertex partition*. Let P_i denote the ith vertex partition of an input graph, i.e., P_i is a subset of V, and $V = \{P_1 \cup P_2 \cup \cdots \cup P_k\}$ $(k \geq 1)$, $\forall P_i, P_j \in V$, $P_i \cap P_j = \emptyset$, $i \neq j$. Our problem is how to add a set of edges such that the resulting graph is 2-edge-connected and the two endpoints of each added edge are not in the same vertex partition.

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 single 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 of 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 makes G 2-edge-connected.

A *block* in a graph is an induced subgraph of a 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*. A *singular* connected component is one formed by an isolated vertex, and a *singular block* is one with exactly one vertex. 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, the blocks $a, b, \cdots, i$ are represented by vertices. The resulting tree is illustrated in Figure 2. A *mono block of P_i* in G is a block comprised of vertices in P_i of G. A *hybrid block* in G is a block containing at least two vertices, one in P_i and another in P_j of G, where $i \neq j$ and $P_i, P_j \in V$. An *isolated mono block of P_i* in G is an isolated block and also a mono block of P_i in G. An *isolated hybrid block* in G is an isolated block and also a hybrid block in G.

The vertices and leaves in BB(G) are defined as follows. Given a graph G with k vertex partitions $P_1, \cdots, P_k$, $k \geq 1$, let C_i denote the ith vertex partition in BB(G), where all corresponding blocks and vertices are in P_i of G. An *isolated vertex of C_i* in BB(G) is an isolated mono block of P_i or an isolated vertex of P_i in G. An *isolated hybrid vertex* in BB(G) is an isolated hybrid block in G. A *mono leaf* (respectively, *mono vertex*) of C_i in BB(G) is a leaf (respectively, vertex) in BB(G), whose corresponding block in G is a mono block of P_i. A *hybrid leaf* in BB(G) is a vertex in BB(G), whose corresponding block in G is a hybrid block.

In addition, let $\mathrm{F_n}$ be a function that can transform an edge set added to BB(G) into a corresponding edge set added to G. If E' is the edge set added to BB(G), then $\mathrm{F_n}(E')$ is the corresponding edge set added to G, i.e., aug2$e(G)$=$\mathrm{F_n}(E')$. Similarly, if e' is an edge added to BB(G), then $\mathrm{F_n}(e')$ is a corresponding edge added between a black vertex and a non-adjacent white vertex of G, if possible.

Given a PRAM model $\mathcal{M}$, let $\mathcal{T}_{\mathcal{M}}(n, m)$ be the parallel time needed to compute the connected components of G using $\mathcal{P}_{\mathcal{M}}(n, m) \leq (n + m)$ processors.

Fact 1 ([1, 2])

1. *If $\mathcal{M} = CRCW$, then $\mathcal{T}_{CRCW}(n, m) = O(\log n)$ and $\mathcal{P}_{CRCW}(n, m) = O((n + m)\cdot\alpha(m, n)/\log n)$.*
2. *If $\mathcal{M} = EREW$, then $\mathcal{T}_{EREW}(n, m) = O(\log n)$ and $\mathcal{P}_{EREW}(n, m) = O(n + m)$.*

A rooted bridge-block forest for a graph can be computed in sequential linear time and in $O(\log n + \mathcal{T}_{\mathcal{M}}(n, m))$ parallel time using $O((n + m)/\log n + \mathcal{P}_{\mathcal{M}}(n, m))$ processors on an $\mathcal{M}$ PRAM [13, 15, 16].

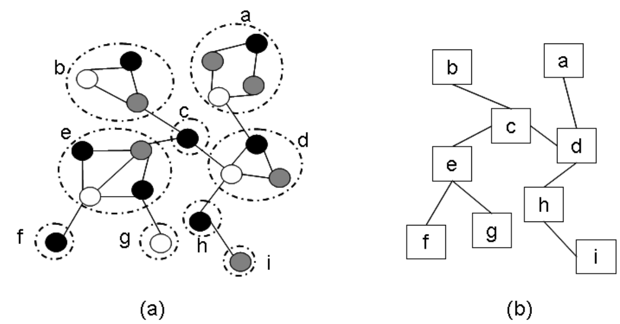

Figure 2: (a) A graph has three vertex partitions and the maximum 2-edge-connected subsets of vertices of this graph are grouped into a set of blocks by the dashed lines. (b) The bridge-block forest of the graph in (a).

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

3 Loose Forest

To reduce the complexity of finding the augmentation of a graph, we introduce a new concept of loose edges. If an edge $e = (u, v)$ with two endpoints u and v in different vertex partitions of G, i.e., $u \in P_i, v \in P_j, i \neq j$, can be removed from the input graph G and its two endpoints u, v can be connected to other vertices in G, then this edge is called a *loose edge*. Conversely, if an edge is not a loose edge, it is called a *fixed edge*. A *loose block* of a graph is the induced subgraph of the maximal 2-edge-connected subset of vertices and contains exactly one loose edge. A bridge-block tree is *loose* if each leaf in the tree is a loose block. A bridge-block forest is defined as a *loose forest*, if all of its trees are loose trees.

In this paper, we first solve the problem of making a loose forest 2-edge-connected. We then solve the problem of a smallest 2-edge-connectivity augmentation of a graph with a partition constraint by transforming the input graph G into a loose forest. For simplifying the discussion, we define two operations *reconnect* and *swap*. *Reconnect* is an operation that removes a set of edges E_r from the given graph or data structure and adds a set of edges E_a, which connect the endpoints of the removed edges, where $|E_r| = |E_a|$ and $E_r \cap E_a = \emptyset$. Let $e_1 = (u_1, v_1)$, $e_2 = (u_2, v_1)$ be two edges, *swap* e_1 and e_2 is an operation that removes e_1, e_2 and adds two edges $e'_1 = (u_1, v_2)$, $e'_2 = (u_1, v_1)$ or $e'_1 = (u_1, u_2)$, $e'_2 = (v_1, v_1)$. The following lemma shows the property of swap operation.

Lemma 1 *(swap property) Given two 2-edge-connected components G_1 and G_2, and two edges e_1, e_2, where $e_1 = (u_1, v_1) \in G_1$, $e_2 = (u_2, v_2) \in G_2$. Let $G' = G_1 \cup G_2 - \{e_1, e_2\} \cup (u_1, v_2) \cup (u_2, v_1)$ or $G' = G_1 \cup G_2 - \{e_1, e_2\} \cup (u_1, u_2) \cup (v_1, v_2)$, then G' is a 2-edge-connected component.*

Proof. After removing e_1 (respectively, e_2), there is still a tree path from u_1 to v_1 in G_1 (respectively, from u_2 to v_2 in G_2). After adding edges (u_1, v_2) and (u_2, v_1) (or (u_1, u_2) and (v_1, v_2)), then G_1 and G_2 are connected, and it is obvious that there exists a cycle containing (u_1, v_1, u_2, v_2). Therefore G' is a 2-edge-connected component. □

Before solving the case of loose forest, we first consider a simpler case that takes a set of loose blocks as an input graph. We propose an method that reconnets a set of loose blocks to a 2-edge-connected component as shown in Algorithm 1.

Algorithm 1 Reconnect a set of loose blocks to a 2-edge-connected component

procedure LBTO2EC(LB, E_L) $\{*$ LB is a set of loose blocks with a set of loose edges E_L in it $*\}$
  $E' = \emptyset$; $E'_L = \emptyset$; $LB' = LB$
  Number each loose block in LB as $b_1, \cdots, b_{|LB|}$;
  Number the loose edge in the loose block b_i as e_i, $1 \leq i \leq |E_L|$; $\{*$ $|LB| = |E_L|$ $*\}$
  if there are at least two loose blocks in LB' **then**
    for i from 1 to $\lfloor E_L/2 \rfloor$ **do**
      $LB' = LB' - e_{2i-1} - e_{2i}$; $\{*$ Assume that $e_{2i-1} = (a, b)$ and $e_{2i} = (c, d)$; $*\}$
      if a and c are in the same vertex partition or b and d are in the same vertex partition **then**
        Let $e'_1 = (a, d)$ and $e'_2 = (b, c)$;
      else
        Let $e'_1 = (a, c)$ and $e'_2 = (b, d)$;
      end if
      Let e'_1 be a loose edge and e'_2 be a fixed edge;
      $LB' = LB' \cup e'_1 \cup e'_2$;
      $E'_L = E'_L \cup e'_1$;
      $E' = E' \cup e'_2$;
    end for
    if $|E_L|$ is odd **then**
      Let $E'_L = E'_L \cup e_{|E_L|}$;
    end if
    Let $E' = E' \cup$ LBTO2EC(LB', E'_L);
  end if
  return E';
end procedure

Lemma 2 *Let* BB(G) *be the input graph and E_L be the set of loose edges in* BB(G). *Then,* BB$(G) - E_L \cup$

E' is a 2-edge-connected compoent, where E' is the edge set that returned by Algorithm 1, and no edges in E' violate the partition constraint.

Proof. By Lemma 1, two loose blocks can be reconnected to a 2-edge-connected component after swapping their loose edges. Note that steps $7-14$ process swap operation between two loose edges. In addition, one of the new edges formed by swap operation in the new block is assigned as a loose edge and the other edge is assigned fixed. The new formed block is also a loose block. Since our algorithm recursively swaps loose edges between two loose blocks until there is only one loose block, the set of loose blocks is reconnected to a 2-edge-connected component.

Now, we consider whether the edges in E' violate the partition constraint. In our algorithm, it swaps two loose edges $e_{2i-1} = (a, b)$, $e_{2i} = (c, d)$ of two loose blocks. Since a loose edge connects two vertices in different partitions, then a (respectively, c) and b (respectively, d) are in different partitions, In addition, it is obvious to see that the partition constraint of these added edges can be guaranteed by steps $8-12$. Therefore, no edges in E' violate the partition constraint. □

Now, we consider a loose forest F as an input graph, it is trivial to see that Algorithm 2 can correctly reconnect a loose forest to a 2-edge-connected component.

Algorithm 2 Reconnect a loose forest to a 2-edge-connected component

```
procedure FTO2EC(F, E_L)   {* F is a loose forest with a set of loose edges E_L. *}
    E' = ∅;
    Let X be a set of leaves and isolated vertices in F;
    E'=LBTO2EC(X,E_L);
    return E';
end procedure
```

4 Main Result

Let G be the input graph. In this paper, we use F and $\text{BB}(G)$ interchangeably to denote the bridge-block forest for an input graph G. $C_1, C_2, \cdots, C_k$ denote the vertex partitions of $\text{BB}(G)$. Let S_i and H denote the sets of mono leaves of C_i and a set of hybrid leaves in $\text{BB}(G)$, respectively. In addition, let S_i^* and H^* denote the sets of isolated vertices of C_i and a set of isolated hybrid vertices in $\text{BB}(G)$, respectively. We say that BB(G) is *C_i-dominated* if $|S_i| + 2|S_i^*| > \lceil(2|S_1^*| + \cdots + 2|S_k^*| + 2|H^*| + |S_1| + \cdots + |S_k| + |H|)/2\rceil$.

Let $|\hat{S_{max}}| = \max\{|S_i|+2|S_i^*| \,|\, 1 \le i \le k\}$. Without loss of generality, we assume that $|S_1|+2|S_1^*| = |\hat{S_{max}}|$.

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

Let $\text{LOW}_{i2e}(\text{BB}(G)) = \max\{|\hat{S_{max}}|, \lceil(2|S_1^*| + \cdots + 2|S_k^*| + 2|H^*| + |S_1| + \cdots + |S_k| + |H|)/2\rceil\}$.

Theorem 1 $|\text{aug2}e(\text{BB}(G))| \ge \text{LOW}_{i2e}(\text{BB}(G))$.

Proof. Note that each leaf needs one incident edge and each isolated vertex needs two incident edges to make the resulting graph 2-edge-connected. Hence, $|\text{aug2}e(G)| \ge \lceil(2|S_1^*| + \cdots + 2|S_k^*| + 2|H^*| + |S_1| + \cdots + |S_k| + |H|)/2\rceil$. By Fact 2, the endpoints of an added edge cannot both be in the same vertex partition. Thus, $|\text{aug2}e(G)| \ge |\hat{S_{max}}|$, so the theorem holds. □

Corollary 1 *If* $\text{BB}(G)$ *is C_i-dominated, then* $\text{LOW}_{i2e}(\text{BB}(G)) = |S_i| + 2|S_i^*|$.

Proof. By definition. □

4.2 Augmentation Algorithm

First, we present an algorithm that numbers the leaves and isolated vertices of an input bridge-block forest, as shown in Algorithm 3. Based on the assigned numbers of leaves and isolated vertices, we can add edges between the leaves and isolated vertices of the input graph without violating the partition constraint. Then, we present an algorithm for finding the augmentation of $\text{BB}(G)$, as shown in Algorithm 4. Here we assume that, $\text{BB}(G)$ contains at least two leaves or two isolated vertices, or at least one leave and one isolated vertex that are in different vertex partitions

Algorithm 3 Numbering the leaves and isolated vertices in F

```
procedure NUMBERING(F)   {* F is a bridge-block forest with k partitions of its vertices; *}
    Let λ_0 = 0;
    for i from 1 to k do
        Assign a number to each leaf in S_i from λ_{i-1}+1 to λ_{i-1} + |S_i|;
        Assign two consecutive numbers to each isolated vertex in S_i^*,
                from λ_{i-1}+|S_i|+1 to λ_{i-1}+|S_i|+2|S_i^*|;
        Let λ_i = λ_{i-1} + |S_i| + 2|S_i^*|;
    end for
    Assign a number to each leaf in H from λ_k + 1 to λ_k + |H|;
    Assign two consecutive numbers to each isolated vertex in H^*, from λ_k + |H| + 1 to λ_k + |H| + 2|H^*|;
end procedure
```

Figure 3 illustrate the numbering procedure in Algorithm 3. In this example, there are three vertex

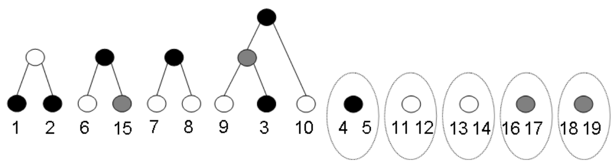

Figure 3: An illustration of the Numbering Procedure.

partitions. The black, white, and gray circles denote the vertices of the first, second, and third partitions, respectively. The black leaves in the graph are numbered 1 to 3 and each isolated vertex is assigned two consecutive numbers; therefore, the black isolated vertices are numbered 4 and 5. Similarly, the vertices of the second and third partitions are assigned consecutive numbers.

Lemma 3 *The partition constraint is maintained in Algorithm 4.*

Proof. According to Algorithm 3, the vertices in the same vertex partition are assigned successive numbers, and the numbers assigned to a vertex partition are equal to $|S_i| + 2|S_i^*|$, $1 \leq i \leq k$. We prove the lemma with the following cases. Note that, $|S_1| + 2|S_1^*| = |S_{\hat{max}}|$. In case 1, $|S_{\hat{max}}|$ is less than $\lfloor \ell/2 \rfloor$. Since $|S_{\hat{max}}| = \max\{|S_i| + 2|S_i^*| | 1 \leq i \leq k\}$, 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.

Case 1.1: ℓ is even. The algorithm only adds an edge between two vertices with numbers i and $i + \ell/2$, $1 \leq i \leq \ell/2$. If two vertices with numbers i and $i + \ell/2$ are in the same vertex partition, then the vertex partition must contain $\ell/2 + 1$ numbers. However, since no vertex partition is assigned more than $\ell/2$ numbers, a vertex partition can not have two vertices with numbers i and $i + \ell/2$.

Case 1.2: ℓ is odd. The algorithm adds an edge between two vertices with numbers i and $i + \lfloor \ell/2 \rfloor$, $1 \leq i \leq \lfloor \ell/2 \rfloor$, and two vertices with numbers $\lfloor \ell/2 \rfloor + 1$ and ℓ. If two vertices with numbers i and $i + \lfloor \ell/2 \rfloor$ or $\lfloor \ell/2 \rfloor + 1$ and ℓ are in the same vertex partition, then the vertex partition must contain $\lfloor \ell/2 \rfloor + 1$ numbers. However, since no vertex partition is assigned more than $\lfloor \ell/2 \rfloor$ numbers, a vertex partition can not have two vertices with numbers i and $i + \lfloor \ell/2 \rfloor$.

Case 2: $|S_{\hat{max}}| > \lfloor \ell/2 \rfloor$. Clearly, the algorithm only adds edges between the vertex partition that has $|S_{\hat{max}}|$ numbers and other vertex partitions. Therefore the lemma holds. □

Figure 4 shows an example that illustrates steps $5 - 15$ of Algorithm 4.

Theorem 2 *Algorithm 4 is correct and optimal.*

Proof. We first prove the correctness of Algorithm 4. In step 5, the algorithm applies Algorithm 3 to assign numbers to leaves and isolated vertices. By Lemma 3, the partition constraint is maintained after adding edges. Steps $16 - 21$, transform a graph into a loose forest. Then, in step 22, the algorithm applies Algorithm 2 to reconnect the loose forest into a single 2-edge-connected component. Therefore, Algorithm 4 is correct.

Next, we prove the optimality of our algorithm, which must consider two cases: case 1, $|S_{\hat{max}}| \leq \lfloor \ell/2 \rfloor$, and case 2. $|S_{\hat{max}}| > \lfloor \ell/2 \rfloor$. It is obvious that the number of edges added in case 1 is equal to $\lceil \ell/2 \rceil$ in steps $8 - 12$ of the algorithm. Therefore, the number of added edges in case 1 is equal to $\mathrm{LOW}_{i2e}(\mathrm{BB}(G))$. Similarly, the number of added edges in case 2 is equal to $|S_{\hat{max}}|$ and $|S_{\hat{max}}| > \lfloor \ell/2 \rfloor$, i.e., $S_{\hat{max}}$-dominated. The number of added edges in case 2 is also equal to $\mathrm{LOW}_{i2e}(\mathrm{BB}(G))$. Note that the algorithm does not add any edges in steps $16 - 20$. Therefore, Algorithm 4 is optimal. □

Theorem 3 *Algorithm 4 runs in sequential linear time and $O(\log n)$ parallel time on an EREW PRAM using a linear number of processors.*

Proof. Given a graph G as input, by Fact 1, the first step in Algorithm 4 takes sequential linear time and $O(\log n + \mathcal{T}_{\mathcal{M}}(n, m))$ parallel time using $O((n+m)/\log n + \mathcal{P}_{\mathcal{M}}(n, m))$ processors on an EREW PRAM to compute BB(G). After computing $\mathrm{BB}(G)$, the numbering procedure takes sequential liner time and $O(\log n)$ parallel time. Then, the algorithm takes $O(1)$ time to determine which case should be executed. In steps $7 - 19$, the algorithm takes sequential liner time and $O(\log n)$ parallel time to add edges between vertices. Finally, the algorithm applies Algorithm 2 to reconnect the graph to a 2-edge-connected component, and it is obvious that Algorithm 2 takes sequential liner time $O(\log n)$ parallel time. Therefore, this theorem holds. □

Note, it is clear that the resulting graph derived by Algorithm 4 is a simple graph, since the algorithm only adds edges between leaves and isolated vertices, and no edge would be added between a vertex and its parent.

5 Concluding remarks

We have proposed a number of algorithms for finding a smallest 2-edge-connectivity augmentation of in-

Algorithm 4 Finding a smallest 2-edge-connectivity augmentation of a graph G with a partition constraint

procedure FS2AUG(G)
    Let $F = \mathrm{BB}(G)$;
    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);
    switch($|\hat{S_{max}}|$)
    Case 1: $|\hat{S_{max}}| \leq \lfloor \ell/2 \rfloor$
        Case 1.1: ℓ is even
            $E' = \{(v_i, v_{i+\ell/2}) | 1 \leq i \leq \ell/2\}$;
        Case 1.2: ℓ 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: $|\hat{S_{max}}| > \lfloor \ell/2 \rfloor$
        $E' = \{(v_i, v_{i+|\hat{S_{max}}|}) | 1 \leq i \leq \ell - |\hat{S_{max}}|\}$;
        $E_1 = \{(v_j, v_\ell) | \ell - |\hat{S_{max}}| + 1 \leq j \leq |\hat{S_{max}}|\}$;
    Let $E' = E' \cup E_1$;
    Let $F' = \mathrm{BB}(F \cup E')$;
    Let X be a set of leaves and isolated vertices in F';
    Arbitrarily select an added edge in each leaf and each isolated vertex of F' from E' and let $E_2 = e_1, e_2, \cdots, e_{|X|}$ denote the set of the selected added edges;
    $E' = E' - E_2$;
    Let all edges in E_2 be loose edges;
    E_2=FTo2EC(F', E_2);
    $E = E' \cup E_2$;
    return E;
end procedure

put graphs with a partition constraint. The proposed methods produce a simple graph if possible, or a multigraph when it is not possible to obtain a simple graph by any approach. The algorithms can be trivially parallelized to run in optimal $O(\log n)$ time using a linear number of EREW processors.

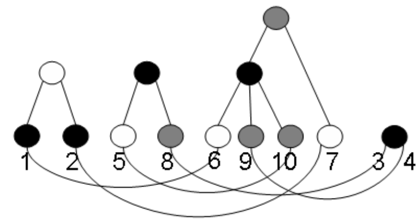

Figure 4: An illustration of steps in Algorithm 4.

References

[1] K. W. Chong, Y Han, and T. W. Lam. Concurrent threads and optimal parallel minimum spanning trees algorithm. *Journal of ACM*, 48(2):297–323, 2001.

[2] R. Cole and U. Vishkin. Approximate parallel scheduling. Part II: Applications to logarithmic-time optimal graph algorithms. *Information and Computation*, 92:1–47, 1991.

[3] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5:653–665, 1976.

[4] A. Frank. Connectivity augmentation problems in network design. In J. R. Birge and K. G. Murty, editors, *Mathematical Programming: State of the Art 1994*, pages 34–63. The University of Michigan, 1994.

[5] F. Harary. *Graph Theory*. Addison-Wesley, Reading, Massachusetts, 1969.

[6] P. C. Huang, H. W. Wei, W. C. Lu, W. K. Shih, and T.-s. Hsu. *Smallest Bipartite Bridge-connectivity Augmentation*. Algorithmica, 2007.

[7] T.-s. Hsu. *Graph Augmentation and Related Problems: Theory and Practice*. PhD thesis, University of Texas at Austin, 1993.

[8] T.-s. Hsu. On four-connecting a triconnected graph. *Journal of Algorithms*, 35:202–234, 2000.

[9] T.-s. Hsu. Simpler and faster biconnectivity augmentation. *Journal of Algorithms*, 45(1):55–71, 2002.

[10] T.-s. Hsu and M. Y. Kao. Optimal augmentation for bipartite componentwise biconnectivity in linear time. *SIAM Journal on Discrete Mathematics*, 19(2):345–362, 2005.

[11] J. B. Jensen, H. N. Gabow, T. Jordan, and Z. Szigeti. Edge-connectivity augmentation with partition constraints. *SIAM Journal on Discrete Mathematics*, 12:160–207, 1999.

[12] H. Nagamochi. Recent development of graph connectivity augmentation algorithms. *IEICE Transactions on Information and System*, E83-D:372–383, 2000.

[13] V. Ramachandran. Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity. In J. H. Reif, editor, *Synthesis of Parallel Algorithms*, pages 275–340. Morgan-Kaufmann, 1993.

[14] A. Rosenthal and A. Goldner. Smallest augmentations to biconnect a graph. *SIAM Journal on Computing*, 6:55–66, 1977.

[15] R. E. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–160, 1972.

[16] R. E. Tarjan and U. Vishkin. An efficient parallel biconnectivity algorithm. *SIAM Journal on Computing*, 14:862–874, 1985.

[17] T. Watanabe and A. Nakamura. A minimum 3-connectivity augmentation of a graph. *Journal of Computer and System Science*, 46:91–128, 1993.