中央研究院
資訊科學研究所
Institute of Information Science, Academia Sinica • Taipei, Taiwan, ROC

TR-IIS-06-016

Smallest Bipartite

Bridge-connectivity Augmentation

Pei-Chi Huang, Hsin-Wen Wei, Wan-Chen Lu, Wei-Kuan Shih and Tsan-sheng Hsu

December 26, 2006 || Technical Report No. TR-IIS-06-016

http://www.iis.sinica.edu.tw/LIB/TechReport/tr2006/tr06.html

Smallest Bipartite Bridge-connectivity Augmentation

Pei-Chi Huang[*], *Hsin-Wen Wei*[*], *Wan-Chen Lu*[*], *Wei-Kuan Shih*[*] *and Tsan-sheng Hsu*[†]

Abstract

This paper addresses two augmentation problems related to bipartite graphs. The first, a fundamental graph-theoretical problem, 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, and still bipartite. The second problem, which arises naturally from research on the security of statistical data, is how to add edges so that the resulting graph is simple and dose not contain any bridges. In both cases, after adding edges, the graph can be either a simple graph or, if necessary, a multi-graph. Our approach then determines whether or not such an augmentation is possible.

We propose a number of simple linear-time algorithms to solve both problems. Given the well-known bridge-block data structure for an input graph, the algorithms 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. We note that there is already a polynomial time algorithm that solves the first augmentation problem related to graphs with a given general partition constraint in $O(n(m + n \log n) \log n)$ time, where m is the number of distinct edges in the input graph. We are unaware of any results for the second problem.

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, the addition of which makes an undirected graph k-edge-connected, is a fundamental problem with many important applications that has been studied extensively; readers may refer to [8, 11, 28] for a comprehensive survey. In

[*]Department of Computer Science, National Tsing-Hua University, Hsinchu, Taiwan. Email: {peggy, bertha, wanchen, wshih}@rtlab.cs.nthu.edu.tw

[†]Institute of Information Science, Academia Sinica, Nankang, Taipei, Taiwan. Email: tshsu@iis.sinica.edu.tw

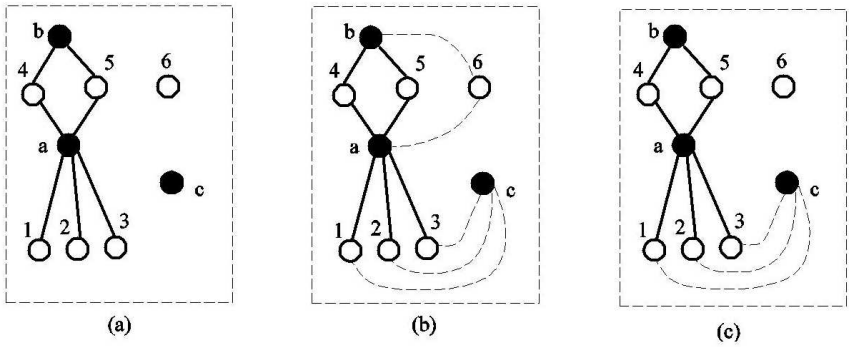

Figure 1: (a) A bipartite graph. The two sets of vertices in the graph are colored black and white, respectively. (b) A smallest 2-edge-connectivity augmentation of (a) with the set of added edges marked by dashed lines. (c) A smallest componentwise 2-edge-connectivity augmentation of (a).

this paper, we focus on augmenting bipartite graphs. A graph is *componentwise 2-edge-connected* if each connected component is either 2-edge-connected, or it is an isolated vertex. We propose a linear-time algorithm that addresses the problem of adding the smallest number of edges to a given bipartite graph to make it 2-edge-connected, or bridge-connected, while maintaining its bipartiteness. We also present another linear-time algorithm that solves the componentwise 2-edge-connectivity augmentation problem in bipartite graphs. Figure 1(a) shows an example of a bipartite graph. A smallest 2-edge-connectivity augmentation of (a) is shown in Figure 1(b), and a smallest componentwise 2-edge-connectivity augmentation of (a) is shown in Figure 1(c). Let G be the input graph and G' be the connected subgraph in G that contains bridges. In the non-bipartite case, it is trivial to know that a smallest 2-edge-connectivity augmentation of G' is equal to a smallest componentwise 2-edge-connectivity augmentation of G. However, this is not true in the bipartite case, as shown in Figure 1.

Note that there is a linear-time algorithm for the smallest bridge-connectivity augmentation problem on the general graph that does not have a bipartite constraint [7]. In [19], Jensen et al. proposed a polynomial time algorithm that solves the smallest bridge-connectivity augmentation problem on a graph that has partition constraints, such as bipartite graph, in $O(n(m+n\log n)\log n)$ time, where m is the number of distinct edges in the input graph. We are unaware of any previous results for the smallest componentwise bridge-connectivity augmentation problem.

1.1 Motivation

Many algorithms have been developed to resolve the problem of adding edges to a graph so that connectivity specifications can be satisfied (e.g., [8, 11, 20]). In addition, the fundamental problem of making general graphs k-edge connected or k-vertex connected for various values of k [7, 12, 13, 14, 15, 18, 30, 33] has been studied extensively (see [28] for examples). Studies of augmentation problems in bipartite graphs can be found in [17, 19, 21].

The related componentwise 2-edge-connectivity augmentation problem arises naturally from research on the security of statistical data [1, 2, 5, 6, 24]. To protect sensitive information in a cross-tabulated table, it is a common practice to suppress some of the cells in the table. A basic issue concerning the effectiveness of this practice is how to suppress a small number of cells, in addition to the sensitive cells, so that the resulting table does not leak important or confidential information. This protection problem can be reduced to an augmentation problem in bipartite graphs [9, 16, 21, 22, 23, 25, 26, 27]. In particular, a linear-time algorithm for the smallest componentwise bipartite 2-edge-connectivity augmentation problem immediately yields a linear-time algorithm that suppresses the smallest number of additional cells so that no nontrivial information about any suppressed cell will be revealed to an adversary [9].

Table 1 and Figure 1(a) illustrate the relationship between our augmentation problem and the table protection problem. Table 1 is a 2-dimensional cross-tabulated table with some suppressed cells. In the bipartite *suppressed graph* constructed from the table, the vertices correspond to the columns and rows, and the edges correspond to the suppressed cells, as shown in Figure 1(a). It has been proven [9] that the value of a suppressed cell can be revealed to an adversary if and only if it is a bridge in the constructed suppressed graph. Therefore, since there are three bridges in our suppressed graph, an adversary could infer the values of the three corresponding cells. For instance, let $C_{i,j}$ be the cell at the intersection of row i and column j, let $S_{*,j}$ be the sum of the cells in column j, and let $S_{i,*}$ be the sum of the cells in row i. Then, the value of $C_{1,a}$ must be 1 because it is equal to $S_{1,*} - C_{1,b} - C_{1,c}$. The value of $C_{5,b}$ is arbitrary. After suppressing three more cells, namely, $C_{1,c}$, $C_{2,c}$, and $C_{3,c}$, the values of the suppressed cells cannot be inferred. This corresponds to the smallest componentwise 2-edge-connectivity augmentation shown in Figure 1(c).

Index	a	b	c	Sum
1		5	2	8
2		3	3	10
3		3	2	12
4			10	20
5			11	24
6	3	4	7	14
Sum	28	25	35	88

Table 1: A 2-dimensional cross-tabulated table with some suppressed cells.

1.2 Our approach and results

We first solve the problem of a smallest 2-edge-connectivity augmentation of bipartite graphs, and then extend the proposed algorithms to deal with the componentwise 2-edge-connectivity case.

To solve the first problem, we transform the input graph G into a well-known data structure called a *bridge-block forest* [10]. A *block* of a graph G is a maximal 2-edge-connected subgraph (or component) of G. We assume B and W are the two bipartite sets of vertices in G. A block that only contains vertices in B (respectively, W) is called a *black* (respectively, *white*) block, while a block that contains both vertices in B and W is called a *hybrid* block. A vertex in the bridge-block forest is *white* if its corresponding block is white. Black and hybrid vertices in the bridge-block forest are defined similarly. Hereafter, we focus on a bridge-block forest, rather than a graph.

Let an *easy tree* be a tree with an equal number of black and white leaves and no hybrid leaves. Our main algorithm first solves the problem on an easy tree, and then solves it on a general tree. Finally, we solve the case where the input graph is a forest. In addition, the edge set added to the bridge-block forest by our algorithms can be transformed into the corresponding edge set added to the input graph G. The algorithms run in sequential liner time and $O(\log n)$ parallel time on an EREW PRAM using a linear number of processors. A high-level description of the algorithm for the 2-edge-connectivity case is given in Algorithm 1.

The remainder of the paper is organized as follows: Section 2 contains graph-theoretical definitions and previously known properties. In Section 3, we propose algorithms that find a smallest 2-edge-connectivity augmentation for a tree. In Section 4, we present the algorithms used when the given input is a forest. In Section 5, we describe our algorithm for finding a smallest

componentwise 2-edge-connectivity augmentation of bipartite graphs. Finally, in Section 6, we present our conclusions.

Algorithm 1 Finding a smallest 2-edge-connectivity augmentation of a bipartite graph G

```
1: procedure FS2Aug(G)
2:     Let T = BB(G);
3:     E = ∅;
4:     repeat
5:         switch (T)
6:         Case 1: T is a tree
7:             Case 1.1: T is an easy tree
8:                 Case 1.1.1: T is an ETC tree
                       E'= ETCT(T);  {* Algorithm 2 *}
9:                 Case 1.1.2: T is an anti-ETC tree with more than 4 leaves
                       E'= AETC(T);  {* Algorithm 3 *}
10:                Case 1.1.3: T is an anti-ETC tree with at most 4 leaves
                       Use the solution shown in Figure 8 to find E';
11:            Case 1.2: T is a general tree
12:                Case 1.2.1: T has no hybrid leaves
                       E'= BGTWAug(T);  {* Algorithm 4 *}
13:                Case 1.2.2: T has hybrid leaves
                       E'= HTAug(T);  {* Algorithm 6 *}
14:        Case 2: T is a forest
15:            Case 2.1: T contains no isolated vertices
16:                Case 2.1.1: T is a light forest with |T_B| = |T_W|
                       E'= FTConversion(T);  {* Algorithm 7 *}
17:                Case 2.1.2: T is a light forest with |T_B| > |T_W|
                       E'= BGTW_FTConversion(T);  {* Algorithm 8 *}
18:                Case 2.1.3: T is a forest with hybrid leaves
                       E'= H_FTConversion(T);  {* Algorithm 9 *}
19:            Case 2.2: T contains a set of isolated vertices S
20:                Case 2.2.1: T − S contains at least 2 white and 2 black vertices
                       Use the method in §4.2.1 to find E';
21:                Case 2.2.2: T − S contains either 1 white or 1 black vertex
                       Use the method in §4.2.2 to find E';
22:                Case 2.2.3: T − S is null
                       E'= ISOF(T);  {* Algorithm 10 *}
23:        Let E = E ∪ E';
24:        Let T = BB(T ∪ E');
25:    until Case 1 is executed
26:    return E;
27: end procedure
```

2 Preliminaries

2.1 Graph-theoretical definitions

In this paper, all graphs are undirected, and have neither self-loops nor multiple edges. Let a graph $G = (V, E)$, where $|V| = n$ and $|E| = m$. Then, for a vertex set V', let $G - V'$ be G without the vertices and their adjacent edges in 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.

An edge whose endpoints are a vertex u and a vertex v is denoted as (u, v). A *bipartite* graph is defined as a graph in which the set of vertices can be partitioned into two disjoint sets such that no edge connects vertices in the same set.

G is a *tree* if it is an undirected, connected, acyclic graph, 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*. Given a rooted tree T and a vertex v of T, let T_v be a proper subtree of T rooted at v. The tree edge that connects the vertex v and its parent is called an *antenna* edge of T_v.

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 $\mathrm{aug2}e(G)$, is a set of edges with the minimum cardinality whose addition makes G 2-edge-connected. A graph is *componentwise 2-edge-connected* if it does not have a bridge. A smallest componentwise 2-edge-connectivity augmentation of G is denoted by $\mathrm{aug}c2e(G)$.

A *block* in a graph is an induced subgraph of a maximal 2-edge-connected subset of vertices. If a block consists of 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. Each bridge in G corresponds to an edge in BB(G) and vice versa. For example, the blocks $a, b, \ldots, i$ are represented by vertices. The resulting tree is illustrated in Figure 2.

In addition, let $\mathrm{F_n}$ be the 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., $\mathrm{F_n}(E')$= aug2$e(G)$. For example,for an input graph and a corresponding bridge-block forest, as shown in Figure 2, assume an edge, e_{added}, is added between vertex a and vertex f, as shown in Figure 3(b). Then, $\mathrm{F_n}$ can transform the edge e_{added} into a corresponding edge e'_{added}, i.e., $\mathrm{F_n}(e_{added})$. The latter is added between an arbitrarily selected black vertex of block a and a white vertex of block f, as shown in Figure 3(a).

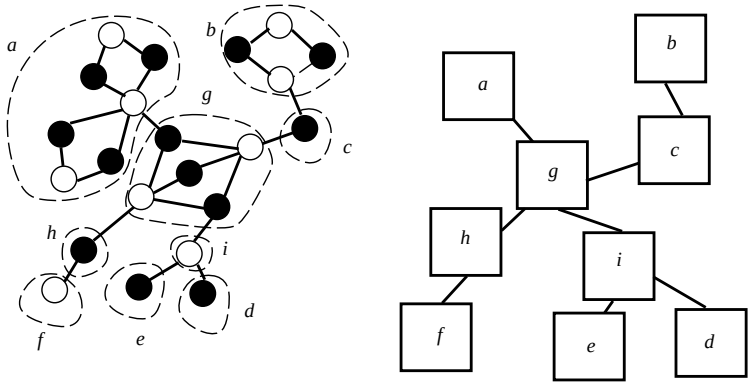

Figure 2: The maximum 2-edge-connected subset of vertices of the bipartite graph on the left are grouped into a set of blocks by dashed lines. The 2-edge forest of the graph is shown on the right.

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 ([3, 4])

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* $\mathcal{P}_{EREW}(n,m) = O(n+m)$.

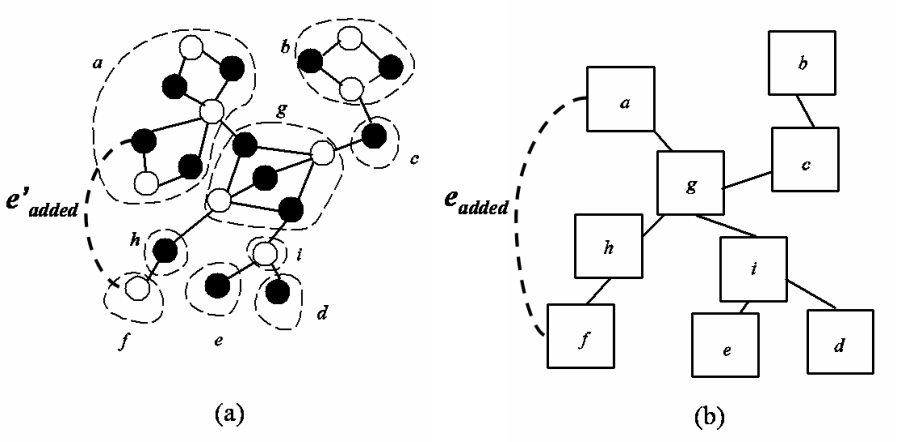

Figure 3: Illustration of the transformation function $\mathrm{F_n}$.

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 [29, 31, 32].

3 Case 1: When BB(G) is a tree

Let G be the input graph. Here, we assume that BB(G) is a tree. Note that, in this paper, we use T and BB(G), interchangeably to denote the bridge-block forest for an input graph G. Since BB(G) is a tree, the implication is that G is connected. Assume BB(G) contains ℓ leaves that can be divided into the following three categories: B is a set of black leaves, W is a set of white leaves, and H is a set of hybrid leaves. Let $|B|$, $|W|$, and $|H|$ be the numbers of black, white, and hybrid leaves in BB(G), respectively. Without loss of generality, we assume that $|B| \geq |W|$. Furthermore, we say that BB(G) is *B-dominated* if $|B| > |W| + |H|$.

3.1 Lower bound on aug2e(BB(G))

Let $\mathrm{LOW}_{t2e}(\mathrm{BB}(G)) = \max\{\lceil(|B| + |W| + |H|)/2\rceil, |B|\}$ when BB(G) is a tree.

Lemma 2 $|\mathrm{aug2}e(\mathrm{BB}(G))| \geq \mathrm{LOW}_{t2e}(\mathrm{BB}(G))$.

Proof. Let E_1 be any set of edges such that BB(G) $\cup$ E_1 is 2-edge-connected. Each leaf must be an endpoint of an added edge in E_1 in order for BB(G) $\cup$ E_1 to be 2-edge-connected. As an edge has two endpoints, we need at least $\lceil(|B| + |W| + |H|)/2\rceil$ edges. Note also that BB(G) is

bipartite; thus, the endpoints of an added edge cannot both be in black leaves. Since we assume that $|B| \geq |W|$, we need at least $|B|$ edges. Therefore, the lemma holds. □

Corollary 3 *If* $\mathrm{BB}(G)$ *is B-dominated, then* $\mathrm{LOW}_{t2e}(\mathrm{BB}(G)) = |B|$.

Proof. By definition. □

3.2 Case 1.1: When $\mathrm{BB}(G)$ is an easy tree

Recall that an easy bridge-block tree T for a bipartite graph is one with an equal number of white and black leaves and no hybrid leaves. We number the leaves of T via a depth-first ordering from 1 to ℓ, i.e., the number of leaves in T, and denote them by $v_1, v_2, \ldots, v_\ell$. Note that, since ℓ is even, $\mathrm{LOW}_{t2e}(\mathrm{BB}(G)) = \ell/2$. By Lemma 2, $|\mathrm{aug2}e(T)| \geq \ell/2$. Our algorithm, described below, always adds $\ell/2$ edges. Thus, after adding edges, if we can prove the resulting graph is 2-edge-connected, the solution found is a smallest 2-edge-connectivity augmentation of T.

If T is an easy tree and there exists i such that v_i and $v_{i+\ell/2}$ are two different-colored leaves, we say that the tree is an *easy-to-connect* or *ETC* tree. An easy tree that is non-ETC is called an *anti-ETC* tree. Note that both ETC and anti-ETC trees are easy trees. Our algorithm considers three cases: (1) an ETC tree, (2) an anti-ETC tree with more than four leaves, and (3) an anti-ETC tree with at most four leaves.

Lemma 4 *Let* T *be the input tree and* $T_{new} = T \cup E_{added}$*, where* E_{added} *is a set of added edges. Then, each added edge* $e \in E_{added}$ *is not a bridge in* T_{new}.

Proof. We prove this lemma by contradiction. Assume the lemma is not true; that is, the added edge e is a bridge. In this case, removing the added edge would increase the number of connected components by one. However, since there was only one connected component before we added this edge, if we remove it, the number of connected components would still be one. Therefore, no new connected component is introduced, which is a contradiction. Thus, Lemma 4 is correct. □

The proof of Lemma 4 is illustrated in Figure 4, in which an added edge connects leaves 3 and 6. Now, since there is a path between the two leaves, it forms a cycle with the added edge. Therefore, the added edge cannot be a bridge.

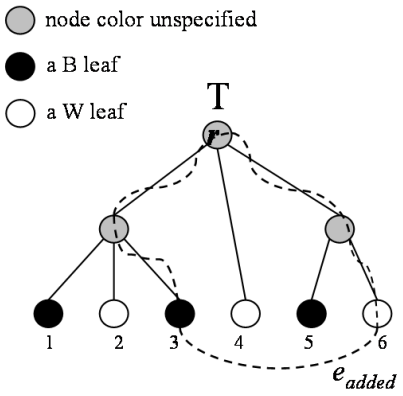

Figure 4: Illustration of the proof of Lemma 4.

3.2.1 Case 1.1.1: When BB(G) is an ETC tree

Our algorithm for finding aug2e(BB(G)) when BB(G) is an ETC tree is shown in Algorithm 2 and illustrated in Figure 5.

Algorithm 2 ETC tree connection

procedure ETCT(T) {$*$ where T is an ETC tree with ℓ leaves $*$}
  Find i^* such that v_{i^*} and $v_{i^*+\ell/2}$ are in different colors;
  Let $V_{in} = \{v_{i^*+1}, v_{i^*+2}, \ldots, v_{(i^*+\ell/2)-1}\}$, and $V_{out} = \{v_1, v_2, \ldots, v_{i^*-1}\} \cup \{v_{(i^*+\ell/2)+1}, v_{(i^*+\ell/2)+2}, \ldots, v_\ell\}$;
  Number the black (respectively, white) leaves in V_{in} starting from 1 as $b_1, b_2, \ldots$ (respectively, $w_1, w_2, \ldots$);
  Number the black (respectively, white) leaves in V_{out} starting from 1 as $b'_1, b'_2, \ldots$ (respectively, $w'_1, w'_2, \ldots$);
  Let $E' = \{(b_i, w'_i) \mid \forall i\} \cup \{(b'_i, w_i) \mid \forall i\}$;
  return $E' \cup \{(v_{i^*}, v_{i^*+\ell/2})\}$;
end procedure

Lemma 5 *For a subtree T' of T, let e_a be the antenna edge of T' and $T_{new} = T \cup E_{added}$, where E_{added} is a set of edges added to T. If there exists an edge $e = (v_a, v_b) \in E_{added}$, such that $v_a \in T'$ and $v_b \notin T'$ or vice versa, then e_a is not a bridge in T_{new}.*

Proof. Before the edge $e = (v_a, v_b)$ is added, there is a path from v_a to v_b in the tree T. Since $v_a \in T'$ and $v_b \notin T'$, the path passes through the antenna edge e_a. After the edge $e = (v_a, v_b)$ has been added, a cycle is formed. However, as the cycle passes through the antenna edge e_a, the latter is not a bridge. □

The proof of Lemma 5 is illustrated in Figure 6, which shows that leaves 7 and 10 are connected. In this case, leaf 7 is in the subtree T', but leaf 10 is not. After the edge connecting the

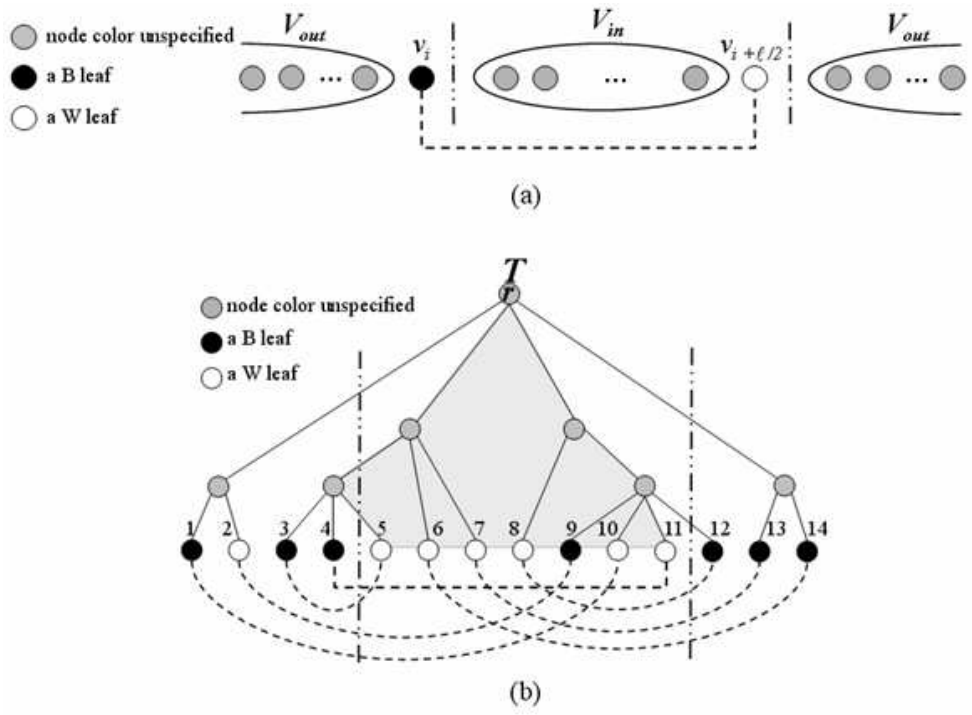

Figure 5: (a) v_i and $v_{i+\ell/2}$ are in different colors with an added edge between them. (b) Illustration of Algorithm 2.

two leaves is added, a cycle is formed such that the antenna edge of T' is in the cycle; therefore, this antenna edge is not a bridge.

Lemma 6 *Let E_{added} be the set of edges derived by Algorithm 2 and let $T_{new} = T \cup E_{added}$. Then T_{new} does not contain any bridge; that is, $E_{added} = \mathrm{aug2e}(T)$.*

Proof. Assume that the lemma is not correct, i.e., there exists a bridge in the resulting graph T_{new}, denoted by e_x. By Lemma 4, no added edge is a bridge; therefore, e_x must be a tree edge. We assume that the subtree corresponding to e_x is T', i.e., e_x is an antenna edge of T', leaves are $v_p, \ldots, v_q$. Since there is an added edge between v_i and $v_{i+\ell/2}$, the possible ranges for p and q are as follows:

Case 1: $1 \le p < q \le i-1$ or $i+\ell/2+1 \le p < q \le \ell$. In this case, all the leaves of T' are in V_{out} and connected to leaves in V_{in} by Algorithm 2. Therefore, by Lemma 5, e_x is not a bridge.

Case 2: $1 \le p \le i$ and $i+1 \le q \le i+\ell/2-1$, or $i+1 \le p \le i+\ell/2-1$ and $i+\ell/2 \le q \le \ell$. In this case, each subtree of T contains a vertex of v_i or $v_{i+\ell/2}$, but a subtree cannot contain both vertices. Since there is an added edge between v_i and $v_{i+\ell/2}$, then, by Lemma 5, e_x is not a bridge.

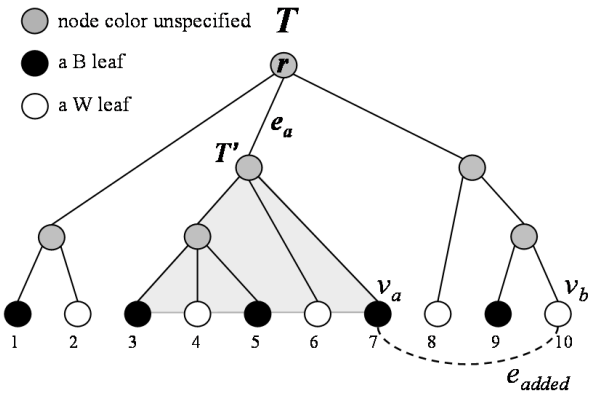

Figure 6: Illustration of the proof of Lemma 5.

Case 3: $i+1 \leq p < q \leq i+\ell/2-1$. In this case, all the leaves of T' are in V_{in} and connected to leaves in V_{out} by Algorithm 2. Therefore, by Lemma 5, e_x is not a bridge.

Case 4: $1 \leq p \leq i-1$ and $i+\ell/2+1 \leq q \leq \ell$. In this case, all leaves in V_{in} are included in the subtree T' and connected to all the leaves in V_{out}. Therefore, since the subtree can not contain all the leaves, by Lemma 5, e_x is not a bridge. □

3.2.2 Case 1.1.2: When BB(G) is an anti-ETC tree with more than four leaves

If a tree is an anti-ETC tree with more than four leaves, we can find two consecutive leaves, denoted, respectively, by v_a and v_{a+1} ($a < \ell/2$), such that v_a and v_{a+1} are different colors. Without loss of generality, we assume that v_a is a black leaf; therefore, v_{a+1} is white. Furthermore, as shown in Figure 7(a) we can find $v_{a+\ell/2}$, which must be black, and $v_{(a+1)+\ell/2}$, which must be white. The steps of the proposed algorithm for this case are given in Algorithm 3 and illustrated in Figure 7.

Algorithm 3 Anti-ETC tree connection

procedure AETC(T) {∗ where T is an anti-ETC tree with ℓ leaves and $\ell > 4$ ∗}
  Find leaves v_a, v_{a+1}, $v_{a+\ell/2}$ and $v_{(a+1)+\ell/2}$ such that v_a and $v_{a+\ell/2}$ are black, and v_{a+1} and $v_{(a+1)+\ell/2}$ are white;
  Let $E_1 = \{(v_a, v_{(a+1)+\ell/2}), (v_{a+1}, v_{a+\ell/2})\}$;
  Let $V_{in} = \{v_{a+2}, v_{a+3}, \ldots, v_{(a+\ell/2)-1}\}$;
  Let $V_{out} = \{v_1, v_2, \ldots, v_{a-1}\} \cup \{v_{(a+1)+\ell/2+1}, v_{(a+1)+\ell/2+2}, \ldots, v_\ell\}$;
  Number the black (respectively, white) leaves in V_{in} starting from 1 as $b_1, b_2, \ldots$ (respectively, $w_1, w_2, \ldots$);
  Number the black (respectively, white) leaves in V_{out} starting from 1 as $b'_1, b'_2, \ldots$ (respectively, $w'_1, w'_2, \ldots$);
  Let $E' = \{(b_i, w'_i) \mid \forall i\} \cup \{(b'_i, w_i) \mid \forall i\}$;
  return $E' \cup E_1$;
end procedure

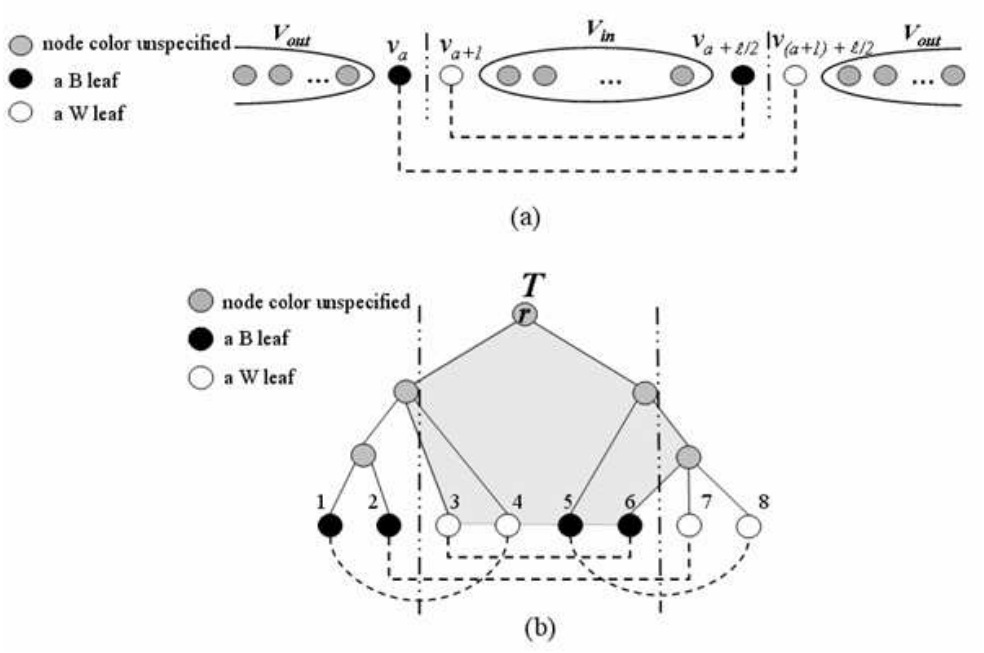

Figure 7: (a) A case where two edges are added. (b) Illustration of Algorithm 3.

Lemma 7 *Let E_{added} be the set of edges derived by Algorithm 3. Then, $T_{new} = T \cup E_{added}$ without bridges.*

Proof. Assume that Lemma 7 is not correct; that is, there exists a bridge , denoted by e_x, in the resulting graph T_{new}. By Lemma 4, e_x is not an added edge. We also assume that the leaves in the subtree T' corresponding to e_x are $v_p, \ldots, v_q$.

Since there are two added edges, one between v_a and $v_{(a+\ell/2)+1}$, and the other between v_{a+1} and $v_{a+\ell/2}$, the ranges of p and q that need to be considered are as follows:

Case 1: $1 \leq p < q \leq a-1$ or $a+\ell/2+2 \leq p < q \leq \ell$. In this case, all the leaves of T' are in V_{out} and connected to leaves in V_{in}. Therefore, by Lemma 5, e_x is not a bridge.

Case 2: $1 \leq p \leq a$ and $a+1 \leq q \leq a+\ell/2$, or $a+1 \leq p \leq a+\ell/2$ and $a+\ell/2+1 \leq q \leq \ell$. In this case, each subtree of T contains some leaves of v_a, v_{a+1}, $v_{a+\ell/2}$, and $v_{a+\ell/2+2}$, but no subtree of T can contain all the leaves. Since there are two added edges between v_a and $v_{(a+\ell/2)+1}$, and between v_{a+1} and $v_{a+\ell/2}$, by Lemma 5, e_x is not a bridge.

Case 3: $a \leq p < q \leq a+\ell/2+1$. In this case, all leaves of T', except v_a, v_{a+1}, $v_{a+\ell/2}$, $v_{a+\ell/2+1}$, are in V_{in} and connected to leaves in V_{out}. Therefore, by Lemma 5, e_x is not a bridge.

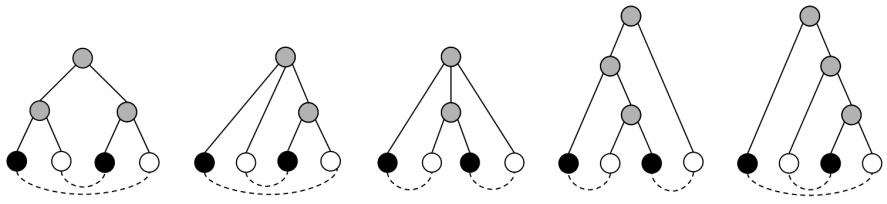

Figure 8: All the possible cases of an anti-ETC tree with exactly four leaves.

Case 4: $1 \leq p \leq a-1$ and $a+\ell/2+2 \leq q \leq \ell$. In this case, all leaves in V_{in} are included in T' and connected to all leaves in V_{out}. Therefore, since T' cannot contain all the leaves, by Lemma 5, e_x is not a bridge. □

3.2.3 Case 1.1.3: When BB(G) is an anti-ETC tree with at most four leaves

Note that an easy tree has an even number of leaves; therefore, a tree can have either two leaves or four leaves in this case. Clearly an easy tree with two leaves must be an ETC tree. Hence, we only need to consider an anti-ETC tree with exactly four leaves. The leaf sequence for such a tree must be colored black, white, black, white, or vice versa. Without loss of generality, we assume the former sequence. Depending on the tree structure, we have the solution for each case of an anti-ETC tree with exactly four leaves, as shown in Figure 8.

3.3 Case 1.2: When BB(G) is a general tree

3.3.1 Case 1.2.1: When BB(G) has no hybrid leaves

Note that, if BB(G) has no hybrid leaves and $|B| = |W|$, then it is an easy tree. Hence, without loss of generality, we assume that $|B| > |W|$. In this case, we apply Algorithm 4.

Lemma 8 *Algorithm 4 is correct.*

Proof. It is straightforward to know that T' computed in step 6 of Algorithm 4 is an easy tree. Hence the correctness of Algorithm 4 follows from Lemmas 6 and 7. □

3.3.2 Case 1.2.2: When BB(G) has hybrid leaves

Note that if an endpoint of an added edge is a hybrid leaf, the other endpoint of the edge can be either black or white. To handle the case when the input is a general tree, we first transform a tree

Algorithm 4 When the input has no hybrid leaves and $|B| > |W|$.

procedure BGTWAUG(T)
  Let b_i be the ith black leaf;
  Let w_i be the ith white leaf;
  Let $V = b_1, b_2, \ldots, b_{|B|}, w_1, w_2, \ldots, w_{|W|}$;
  Let $V' = b_{|W|+1}, b_{|W|+2}, \ldots, b_{|B|}$;
  Let $T' = T - V'$;
  if T' is an ETC tree **then**
    E_1=ETCT(T'); {∗ Algorithm 2 ∗}
  else if T' is an anti-ETC tree with more than 4 leaves **then**
    Let E_1=AETC(T'); {∗ Algorithm 3 ∗}
  else if T' is an anti-ETC tree with at most 4 leaves **then**
    Use the solution illustrated in Figure 8 to find E_1;
  end if
  if there is only one white vertex in T i.e., there is no white leaf in T **then**
    Let u be the white vertex in T;
    $E_2 = \{(b_i, u) \mid 1 \leq i \leq |B|\}$;
  else
    Let u_1, u_2 be two white vertices in T;
    Let $E_2 = \{(b_i, u_j) \mid |W+1| \leq i \leq |B|, j \in \{1, 2\}$, where u_j is not the neighbor of $b_i\}$; {∗ add edges between a white vertex and the remaining black leaves ∗}
  end if
  return $E_1 \cup E_2$;
end procedure

with hybrid leaves into a tree without hybrid leaves using an algorithm called HASSIGN, described in Algorithm 5. Then, we apply Algorithm 4 to the recolored tree derived by Algorithm 5. The steps followed in this case are described in Algorithm 6.

Algorithm 5 H assignment

procedure HASSIGN(T) {∗ where T is a tree with hybrid leaves ∗}
  if $|B| > \lceil(|B| + |W| + |H|)/2\rceil$ **then**
    All hybrid leaves are recolored white;
  else
    Arbitrarily select $|B| - |W|$ hybrid leaves to be recolored white;
    The remaining $\lfloor(|H| - |B| + |W|)/2\rfloor$ hybrid leaves are recolored white;
    The rest are recolored black;
  end if
  Let T' be the resulting tree;
  return T';
end procedure

Lemma 9 *Algorithm 6 is correct and optimal. That is,* $\text{aug2}e(T) = \text{aug2}e(T')$*, where* $T = \text{BB}(G)$ *and* T' *is the recolored tree returned by Algorithm 5.*

Proof. First, we prove the correctness of the algorithm. Let T' be the recolored tree returned by Algorithm 5. We apply Algorithm 4 to T', which, by Lemma 8, guarantees that the black leaf will

Algorithm 6 When T has hybrid leaves

```
1: procedure HTAug(T)
2:     T'= HAssign(T);  {* Algorithm 5 *}
3:     E'= BGTWAug(T');  {* Algorithm 4 *}
4:     return E';
5: end procedure
```

be connected to the white leaf in T'. Algorithm 5 only assigns the hybrid leaves in T to be black or white leaves, so the tree structure of T remains unchanged. Therefore, an edge added to T' can also be added to T, i.e., $\text{aug2}e(T) = \text{aug2}e(T')$.

Next, we prove the optimality of Algorithm 6.

Without loss of generality, we assume that $|B| \geq |W|$.

Case 1: $|B| > |W| + |H|$. In this case, the tree is B-dominated. After applying Algorithm 5, all hybrid leaves are recolored white. However, the tree is still B-dominated and the number of added edges is equal to $\text{LOW}_{t2e}(T)$, i.e., $|B|$.

Case 2: $|B| \leq |W| + |H|$. After applying Algorithm 5, the number of black leaves is equal to $\lceil \ell/2 \rceil$. In this case, if the tree is not B-dominated, $\lceil \ell/2 \rceil$ edges are added such that the number of added edges is equal to $\text{LOW}_{t2e}(T)$.

Therefore, Algorithm 6 is correct and optimal. □

Lemma 10 *$G \cup \text{F}_\text{n}(\text{aug2}e(\text{BB}(G)))$, where $\text{aug2}e(\text{BB}(G))$, found by Algorithm 6, is simple if and only if G has at least two white and two black vertices.*

Proof. Algorithm 6 returns a simple graph, unless steps 14-16 in Algorithm 4 are executed. In the latter case, G would be a star with exactly one black or white vertex. It is straightforward to see that such a graph G dose not have a simple 2-edge-connectivity augmentation. □

4 Case 2: When $\text{BB}(G)$ is a forest

In this section, we describe the remainder of our main algorithm when the input graph G is not connected; in other words, when $\text{BB}(G)$ is a forest instead of a tree.

Recall that a leaf in a forest is a degree-1 vertex; and B, W, and H are, respectively, the sets of black, white, and hybrid leaves in $\text{BB}(G)$. Without loss of generality, we assume that $|B| \geq |W|$.

Let B', W', and H' be, respectively, the sets of isolated black, white, and hybrid vertices in $\mathrm{BB}(G)$. We now present a simple lower bound for $|\mathrm{aug2}e(\mathrm{BB}(G))|$.

Let $\mathrm{LOW}_{f2e}(\mathrm{BB}(G)) = \max\{2|B'| + |B|, 2|W'| + |W|, \lceil(2|B'| + 2|W'| + 2|H'| + |B| + |H| + |W|)/2\rceil\} = p + \max\{|B| + |B'| - |W'| - |H'|, |W| + |W'| - |B'| - |H'|, \lceil(|B| + |H| + |W|)/2\rceil\}$, where p is the number of isolated vertices in $\mathrm{BB}(G)$. Note that if $\mathrm{BB}(G)$ is a tree, $\mathrm{LOW}_{t2e}(\mathrm{BB}(G)) = \mathrm{LOW}_{f2e}(\mathrm{BB}(G))$.

Lemma 11 $|\mathrm{aug2}e(\mathrm{BB}(G))| \geq \mathrm{LOW}_{f2e}(\mathrm{BB}(G))$.

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, $|\mathrm{aug2}e(\mathrm{BB}(G))| \geq \lceil(2|B'| + 2|W'| + 2|H'| + |B| + |H| + |W|)/2\rceil$. Since the graph is bipartite, the endpoints of an added edge cannot both be black or white. Thus, $|\mathrm{aug2}e(\mathrm{BB}(G))| \geq 2|B'| + |B|$ and $|\mathrm{aug2}e(\mathrm{BB}(G))| \geq 2|W'| + |W|$. Note also that $p = |B'| + |W'| + |H'|$, so the lemma holds. □

4.1 Case 2.1: When $\mathrm{BB}(G)$ contains no isolated vertices

In this subsection, we assume that $\mathrm{BB}(G)$ dose not have any isolated vertices. Here, $\mathrm{BB}(G)$ is a forest consisting of non-trivial trees. Note that if a tree is not trivial, i.e., it has only one vertex, then it contains at least two leaves. We propose algorithms that add a set of edges, E_{added}, so that $\mathrm{BB}(G) \cup E_{added}$ is a tree and $\mathrm{aug2}e(\mathrm{BB}(G)) - E_{added} = \mathrm{aug2}e(\mathrm{BB}(\mathrm{BB}(G) \cup E_{added}))$.

First, we consider the case where $\mathrm{BB}(G)$ dose not contain any hybrid leaves. This is called a *light forest*; otherwise, it is called a *general forest*. In a light forest, the trees can be classified into three different types: (1) T_B={ T | T, a tree with only black leaves in T}; (2) T_W={ T | T, a tree with only white leaves in T}; and (3) T_{BW}={ T | T, a tree with at least one black and one white leaf in T}.

Without loss of generality, we assume that $|T_B| \geq |T_W|$. Next, we propose algorithms for two cases: (1) $|T_B| = |T_W|$, and (2) $|T_B| > |T_W|$. For the remainder of this section, let $F = \mathrm{BB}(G)$.

4.1.1 Case 2.1.1: When $\mathrm{BB}(G)$ is a light forest and $|T_B| = |T_W|$

Let $k = |T_B| = |T_W|$ and $z = |T_{BW}|$. Algorithm 7 adds a set of edges to F so that the resulting graph is a tree, as shown in Figure 9.

Algorithm 7 Forest-Tree Conversion

1: **procedure** FTConversion(F) {$*$ F is a light forest with $|T_B| = |T_W| = k$ and $|T_{BW}| = z$ $*$}
2: Number each tree in T_B as 1, 3, $\ldots$, $2k-1$;
3: Number each tree in T_W as 2, 4, $\ldots$ $2k$;
4: Number each tree in T_{BW} as $2k+1$,$2k+2$, $\ldots$ $2k+z$;
5: Give two labels to each tree as follows:
6: **for** the tree i from 1 to $2k+z$ **do**
7: **if** the tree $\in T_B$ **then**
8: Assign the labels $2i-1$ and $2i+1$ to two leaves chosen arbitrarily;
9: **else if** the tree $\in T_W$ **then**
10: Assign the labels $2i-2$ and $2i$ to two leaves chosen arbitrarily;
11: **else if** the tree $\in T_{BW}$ **then**
12: Assign the labels $2i-1$ and $2i$ to two different colored leaves. Here, $2i-1$ is assigned to the black leaf and $2i$ is assigned to the white leaf;
13: **end if**
14: **end for**
15: $E_1 = \{(v_{2j}, v_{2j+1})$ |for all labeled leaves v_{2j} and v_{2j+1}, $1 \le j \le 2k+z-1\}$;
16: **return** E_1;
17: **end procedure**

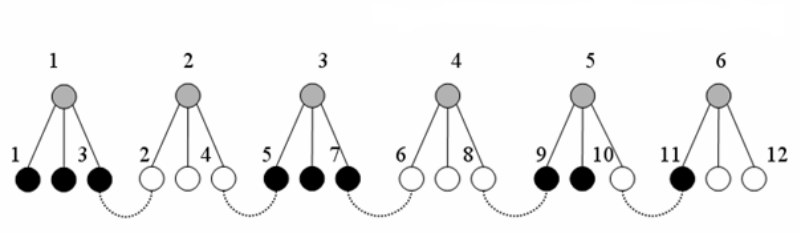

Figure 9: Illustration of Algorithm 7.

In steps 2 – 4 of Algorithm 7, labels in the form of integers are assigned to all the trees in F. Meanwhile, in steps 6 – 13, labels in the form of integers are assigned to some leaves of F. In the following lemmas, we describe the properties of these labels. Then, we demonstrate the correctness of our algorithm based on the following lemmas.

Lemma 12 *The leaves with odd labels are black and the leaves with even labels are white.*

Proof. By definition. □

Lemma 13 *The labels assigned to each tree of the input forest are distinct.*

Proof. Consider any two trees in T_B with numbers x and y respectively, where $|x-y| \ge 2$. Since the labels assigned to the tree numbered x are $2x-1$ and $2x+1$, and the labels assigned to the tree numbered y are $2y-1$ and $2y+1$, the labels assigned to the trees are different. This is also true for trees in T_W. Note that, by definition, trees in T_B and T_W are not assigned the same numbers.

The number of black trees is equal to the number of white trees; hence, the labels assigned to trees in T_B and T_W are in the range $(1, 4k)$, while the labels assigned to trees in T_{BW} are in the range $(4k+1, 4k+2z)$. As these two ranges do not overlap, the lemma is correct. □

Lemma 14 *For each q in the range $1 \leq q \leq 2(2k+z)-1$, there is a path between the leaf with label q and the leaf with label $q+1$.*

Proof. We prove this lemma according to the position of the leaf with label q, as shown by the following three cases.

1. If there is a black leaf in T_B:
   (a) If $q = 2i-1$, then there is a tree path from leaf q to leaf $2i+1$. Since there is also an added edge connecting leaf $2i+1$ and leaf $2i$, there is also a path between leaf $q = 2i-1$ and $q+1$. Note that $q+1 = 2i$.
   (b) If $q = 2i+1$, then there is an added edge connecting leaf $2i+1$ and leaf $2i$. As there is also a tree path from leaf $2(i+1)-2$ to leaf $2(i+1)$, and $2i+2 = q+1$, there is a path between leaf q and $q+1$.
2. If there is a white leaf in T_W:
   (a) If $q = 2i-2$, then there is an added edge between leaf $q = 2(i-1)$ and leaf $q+1 = 2(i-1)+1$.
   (b) The same is true for $q = 2i$.
3. If there is a leaf in T_{BW}:
   (a) If $q = 2i-1$, there is a tree path between $q = 2i-1$ and $2i$.
   (b) If $q = 2i$, there is an added edge connecting $q = 2i$ and leaf $2i+1$.

Therefore, there is a path from leaf q to $q+1$. □

Theorem 15 *$F \cup E_{added}$ is a tree, where E_{added} is the set of edges returned by Algorithm 7. Furthermore, $|\mathrm{LOW}_{f2e}(\mathrm{BB}(G))| = |\mathrm{LOW}_{f2e}(\mathrm{BB}(\mathrm{BB}(G) \cup E_{added}))| + |E_{added}|$.*

Proof. By Lemmas 12, 13, and 14. □

Algorithm 8 $|T_B| > |T_W|$ Forest-Tree Conversion

1: **procedure** BGTW_FTConversion(F) {$*$ where F is a light forest with $|T_B| = k + x$, $(x \geq 1)$, $|T_W| = k$, and $|T_{BW}| = z$ $*$}
2: **if** $T_{BW} = T_W = \phi$ **then**
3: Pick a leaf from each tree in T_B and number them as b_1, b_2, ... ,b_{k+x};
4: Let $E_1 = \{(b_i, u) \mid 1 \leq i \leq k + x$ and let u be a white vertex of T_B with the number b_r, where $i \neq r$ and $1 \leq r \leq k + x\}$;
5: **else**
6: Find a subset T'_B of T_B, such that $|T'_B| = k$;
7: Let $\overline{T'_B} = T_B - T'_B$ and $|\overline{T'_B}| = x$;
8: Let $F' = T'_B \cup T_W \cup T_{BW}$;
9: E_1= FTConversion(F'); {$*$ Algorithm 7 $*$}
10: Pick a leaf from each tree in $\overline{T'_B}$ and number them as b_1, b_2, ..., b_x;
11: Number the remaining white leaves of BB($F' \cup E_1$) as w_1, w_2, ,... w_y; {$*$ assuming there are y remaining white leaves $*$};
12: **if** $x \leq y$ **then**
13: Let $E_2 = \{(b_i, w_i) \mid 1 \leq i \leq x\}$;
14: **else**
15: Let $E_2 = \{(b_i, w_i) \mid 1 \leq i \leq y\} \cup \{(b_i, u) \mid \ y < i \leq x, u$ is an arbitrary white leaf.$\}$;
16: **end if**
17: **end if**
18: **return** $E_1 \cup E_2$;
19: **end procedure**

4.1.2 Case 2.1.2: When BB(G) is a light forest and $|T_B| > |T_W|$

The steps for this case are shown in Algorithm 8, and illustrated in Figure 10.

Theorem 16 *$F \cup E_{added}$ is a tree in which E_{added} is the set of edges derived by Algorithm 8.*

Proof. Let $F = \{T_B, T_W, T_{BW}\}$ be the input forest, and let $T_B = T'_B + \overline{T'_B}$ and $|T'_B| = |T_W|$. Let $F' = T'_B \cup T_W \cup T_{BW}$. Note that E_1 is the set of edges derived by Algorithm 7 using the input F' in step 9. By Theorem 15, $T' = F' \cup E_1$ is a tree. Note that, each tree T^* in $\overline{T'_B}$ has one label assigned to its leaf. Furthermore, it is obvious that there is an edge e in the set of edges E_2 found in steps 12 – 16 such that one endpoint of e is in T^* and the other is in T'. Therefore, $F \cup E_{added}$ is a tree. □

Theorem 17 $|\text{LOW}_{f2e}(\text{BB}(G))| = |\text{LOW}_{f2e}(\text{BB}(\text{BB}(G) \cup E_{added}))| + |E_{added}|$, *where E_{added} is the set of added edges derived by Algorithm 8.*

Proof. Let $q = |E_{added}|$. The two endpoints of each added edge are leaves in BB(G). Let $T' = \text{BB}(G) \cup E_{added}$. By Theorem 16, T' is a tree, therefore, i.e., BB(T') = T'. Each leaf in BB(G) that is an endpoint of E_{added} becomes an interior node in T'. In T', let B_{new} and W_{new}

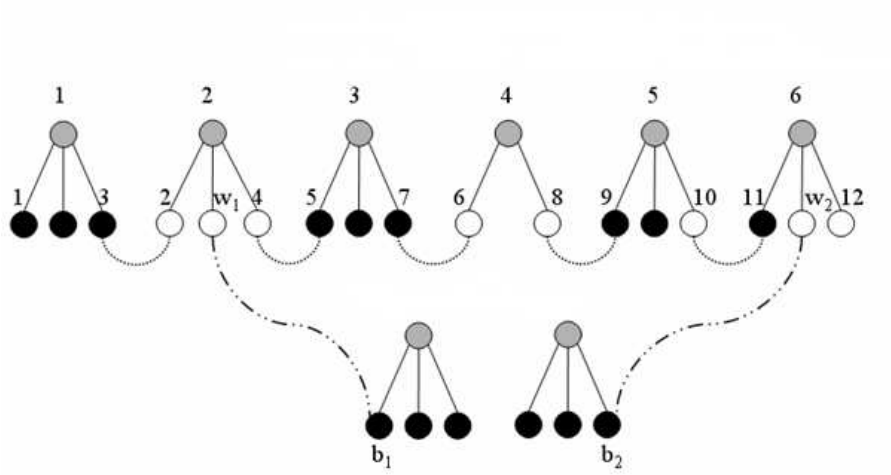

Figure 10: Illustration of Algorithm 8.

denote the black and white leaves, respectively. Then, $|B| = |B_{new}| + q, |W| = |W_{new}| + q$. Hence, the theorem holds. □

In Sections 4.1.1 and 4.1.2, we presented two algorithms that convert a light forest into a tree. If all the leaves of a forest are either black or white, the forest can be transformed into a tree by applying the algorithms presented in this subsection. After this transformation, we can apply the algorithms presented in Section 3 to add edges such that no bridges exist in the final graph.

4.1.3 Case 2.1.3: When BB(G) has hybrid leaves

If one endpoint of an added edge is a hybrid leaf, the other endpoint of that edge can be either black or white. For a general forest, we first transform some trees with hybrid leaves into trees without hybrid leaves using Algorithm 5, the leaf-recoloring algorithm. Then, we apply Algorithm 8 to convert a forest into a tree. The steps followed in this case are shown in Algorithm 9.

Algorithm 9 When the input is a forest that has hybrid leaves

```
1: procedure H_FTConversion(F)  {* where F is a forest with hybrid leaves *}
2:     T'= HAssign(F);  {* Algorithm 5 *}
3:     E'=BGTW_FTConversion(T')  {* Algorithm 8 *}
4:     return E';
5: end procedure
```

Lemma 18 *Algorithm 9 finds* aug2e(BB(G)) *when* BB(G) *is a forest containing no isolated vertices.*

Proof. First, we prove the correctness of the algorithm. Let T' be the recolored forest returned by Algorithm 5. Note that Algorithm 8 guarantees a black leaf will connect to a white leaf in T'. Since the HAssign algorithm only assigns the hybrid leaves in F to be black or white and the tree structure of F remains unchanged, an edge added to T' can also be added to F, i.e., $\text{aug2}e(F) = \text{aug2}e(T')$.

Next, we prove the optimality of the algorithm. Assume F contains ℓ leaves that can be divided into the following three categories: $|B|$ - the number of black leaves, $|W|$ - the number of white leaves, and $|H|$ - the number of hybrid-colored leaves. Without loss of generality, we assume that $|B| \geq |W|$.

Case 1. $|B| > |W| + |H|$

In our algorithm, after applying the HAssign algorithm, all hybrid leaves are assigned as white leaves. Let $|W_{new}|$ be the resulting number of white leaves. Since $|W_{new}| = |W| + |H| < |B|$, and $\lceil (|B| + |W_{new}|)/2 \rceil < |B|$, then $\text{LOW}_{f2e}(T) \geq \text{LOW}_{f2e}(T')$, even if we recolor all the hybrid leaves white. Therefore, our algorithm is optimal.

Case 2. $|B| \leq |W| + |H|$

After applying Algorithm 9, the number of black leaves is equal to $\lceil \ell/2 \rceil$. When the number of black leaves in the forest is $\lceil \ell/2 \rceil$, our algorithms add $\lceil \ell/2 \rceil$ edges. Since the total number of added edges is equal to $\text{LOW}_{f2e}(T)$, our algorithm is optimal. □

4.2 Case 2.2: When BB(G) contains isolated vertices

Recall that each isolated black (respectively, white) block is an isolated black (respectively, white) vertex in G. Let b'_i (respectively, w'_i) be the ith isolated black (respectively, white) vertex in G, and let $h'_{1,i}$, $h'_{2,i}$ be arbitrary black and white vertices, respectively, in the ith isolated hybrid block of G.

Let G' be the graph obtained by removing the vertices and edges from the isolated blocks of G. There are three cases, which we describe below.

- Case 2.2.1: G' contains at least two white and two black vertices. Without loss of generality, we assume that $|B'| \geq |W'|$, which yields five sub-cases:
  - Case 2.2.1.1: $|W'| > 0$.

  - Case 2.2.1.2: $|W'| = 0$, $|B'| > 0$ and $|H'| > 0$.
  - Case 2.2.1.3: $|W'| = 0$, $|B'| = 0$ and $|H'| > 0$.
  - Case 2.2.1.4: $|W'| = 0$, $|B'| > 0$, $|H| + |W| > 0$, and $|H'| = 0$.
  - Case 2.2.1.5: $|W'| = 0$, $|B'| > 0$, $|H| + |W| = 0$, and $|H'| = 0$.
- Case 2.2.2: G' contains either a white or a black vertex. Without loss of generality, we assume that G' contains exactly one white vertex, which yields two sub-cases:
  - Case 2.2.2.1: there is no white vertex in $G - G'$.
  - Case 2.2.2.2: there is a white vertex in $G - G'$.
- Case 2.2.3: G' is null.

4.2.1 Case 2.2.1: G' has at least two white and two black vertices

Without loss of generality, we assume that $|B'| \geq |W'|$. Let E' be the set of added edges to be decided in each sub-case; $\hat{T} = \text{BB}(\text{BB}(G) \cup E')$; $\hat{B}'$, $\hat{W}'$, and $\hat{H}'$ be the respective sets of isolated black, white, and hybrid blocks in $\hat{T}$; and $\hat{B}$, $\hat{W}$, and $\hat{H}$ be the respective sets of black, white, and hybrid leaf-blocks in $\hat{T}$.

Case 2.2.1.1: $|W'| > 0$. Since we assume that $|B'| \geq |W'|$, $|B'| > 0$, let $E' = \{(b'_i, w'_i) \mid 1 \leq i \leq |W'|\}$. Then, $|\hat{B}| = |B| + |W'|$, $|\hat{W}| = |W| + |W'|$, $\hat{H} = H$, $|\hat{W}'| = 0$, $|\hat{B}'| = |B'| - |W'|$, and $\hat{H}' = H'$. Thus, $\text{LOW}_{f2e}(\hat{T}) \geq \text{LOW}_{f2e}(\text{BB}(G)) - |W'|$. We have reduced Case 2.2.1.1 to Case 2.2.1.2, Case 2.2.1.3, Case 2.2.1.4, Case 2.2.1.5 or Case 2.1.

Case 2.2.1.2: $|W'| = 0$, $|B'| > 0$ and $|H'| > 0$. Let $k = \min\{|B'|, |H'|\}$ and $E' = \{(b'_i, h'_{2,i}) \mid 1 \leq i \leq k\}$. Then, $|\hat{B}| = |B| + k$, $\hat{W} = W$, $|\hat{H}| = |H| + k$, $|\hat{W}'| = 0$, $|\hat{B}'| = |B'| - k$, and $|\hat{H}'| = |H'| - k$. Thus, $\text{LOW}_{f2e}(\hat{T}) \geq \text{LOW}_{f2e}(\text{BB}(G)) - k$. We have reduced Case 2.2.1.2 to Case 2.2.1.3, Case 2.2.1.4, Case 2.2.1.5 or Case 2.1.

Case 2.2.1.3: $|W'| = 0$, $|B'| = 0$, and $|H'| > 0$. Let w, h, and b_i be, respectively, arbitrary white, hybrid, and ith black leaves in $\text{BB}(G')$ if they exist. Let $k = \min\{|B|, |H'|\}$ and $E_1 = \{(b_i, h'_{2,i}) \mid 1 \leq i \leq k\}$. If $|H'| > k$, then let $|H''| = |H'| - k$ and $E_2 = \{(h'_{1,i}, h'_{2,i+1})) \mid 1 \leq i < \lfloor |H''|/2 \rfloor\}$. Furthermore, when $|H''|$ is odd, let $E_3 = \{(h'_{1,|H'|}, w)\}$ if w exists; otherwise,

$E_3 = \{(h'_{1,|H'|}, h)\}$. Then, $|\hat{B}| = |B| - k$, $|\hat{H}| = |H| + k + |H''|$, $|\hat{H}'| = 0$, and $E' = E_1 \cup E_2 \cup E_3$. Thus, $\text{LOW}_{f2e}(\hat{T}) \geq \text{LOW}_{f2e}(\text{BB}(G)) - |E'|$. We have reduced Case 2.2.1.3 to Case 2.1.

Case 2.2.1.4: $|W'| = 0$, $|B'| > 0$, $|H| + |W| > 0$, and $|H'| = 0$. Let $k = \min\{|B'|, |H| + |W|\}$; w_i be a white vertex in the ith leaf of $H \cup W$, and $E' = \{(b_i, w_i) \mid 1 \leq i \leq k\}$. Then, $|\hat{B}| = |B| + |k|$, $|\hat{W}| + |\hat{H}| = |H| + |W| - k$, $|\hat{W}'| = 0$, $|\hat{B}'| = |B'| - k$, and $|\hat{H}'| = 0$. Thus, $\text{LOW}_{f2e}(\hat{T}) \geq \text{LOW}_{f2e}(\text{BB}(G)) - k$, so we have reduced Case 2.2.1.4 to either Case 2.2.1.5 or Case 2.1.

Case 2.2.1.5: $|W'| = 0$, $|B'| > 0$, $|H| + |W| = 0$, and $|H'| = 0$. Now, we only have black leaves and isolated black vertices. Let w be a white vertex in G', and $E' = \{(b'_i, w) \mid 1 \leq i \leq |B'|\}$. Note that, in this case, $2|B'| + |B| > \lceil (2|B'| + 2|W'| + 2|H'| + |B| + |H| + |W|)/2 \rceil$. Therefore, $\hat{B} = B \cup B'$, $\hat{W} = \emptyset$, $\hat{H} = \emptyset$, $\hat{W}' = \emptyset$, $\hat{B}' = \emptyset$, and $\hat{H}' = \emptyset$, such that $\text{LOW}_{f2e}(\hat{T}) \geq \text{LOW}_{f2e}(\text{BB}(G)) - |E'|$. We have reduced Case 2.2.1.5 to Case 2.1.

4.2.2 Case 2.2.2: G' contains either one white or one black vertex

Without loss of generality, we assume that G' consists of exactly one white vertex w. Hence, $|H| = 0$ and G' is a star with center w; $\text{BB}(G)$ is also a star. There are two sub-cases: (1) $G - G'$ contains a white vertex, and (2) $G - G'$ does not contain a white vertex.

Case 2.2.2.1: there is no white vertex in $G - G'$. All isolated vertices in G are black, $|W'| = 0$ and $|H'| = 0$, such that $|\text{LOW}_{f2e}(\text{BB}(G'))| = |B|$. Let $E' = \{(b'_i, w) \mid \forall i\}$.

Lemma 19 *For Case 2.2.2.1,* $\text{aug2}e(\text{BB}(G)) = \text{aug2}e(\text{BB}(G')) \cup E'$, *and* $\text{BB}(G) \cup \text{aug2}e(\text{BB}(G))$ *is a multi-graph.*

Proof. By Lemmas 10 and 11. □

Case 2.2.2.2: there is one white vertex in $G - G'$. Let b be a black leaf in $\text{BB}(G')$. Since $\text{BB}(G')$ is a star with a white center, b must exist. Let w' be a white vertex in an isolated block (i.e., in $G - G'$), and let $G'' = \text{BB}(G') \cup \{(w', b)\}$. Note that the number of isolated blocks in $\text{BB}(G) \cup \{(w', b)\}$ is one less than in $\text{BB}(G)$, and the number of black leaves in $\text{BB}(G'')$ is one less than in $\text{BB}(G')$. However, there is one more white leaf in $\text{BB}(G'')$ than in $\text{BB}(G')$. Thus, we have transformed Case 2.2.2.2 into Case 2.2.1.

Lemma 20 *For Case 2.2.2.2,* $|\text{LOW}_{f2e}(\text{BB}(\text{BB}(G) \cup \{(w', b)\}))| = |\text{LOW}_{f2e}(\text{BB}(G))| - 1$.

Algorithm 10 When G' is null, i.e., $\mathrm{BB}(G)$ consists of isolated vertices

procedure ISOF(F) {$*$ where F is a forest that consists of isolated vertices S $*$}
  if $q_B = 0$ **then** {$*$ q_H must be at least 2; $*$}
    Let $E_1 = \{(h'_{1,2i-1}, h'_{2,2i}) \mid 1 \le i \le \lfloor q_H/2 \rfloor\}$;
    if q_H is odd number **then**
      $E' = E_1 \cup (h'_{1,q_H-1}, h'_{2,q_H})$;
    end if
  else
    if $q_B > q_W + q_H$ **then**
      Let $E_1 = \{(b'_i, w'_i) \mid 1 \le i \le q_W\} \cup \{(b'_{i+q_W}, h'_{2,i}) \mid 1 \le i \le q_H\}$;
      $E' = E_1 \cup \{(b'_{i+q_W+q_H}, w) \mid 1 \le i \le q_B - q_W - q_H$ and w is a white vertex in $S\}$;
    else if $q_B = q_W + q_H$ **then**
      Let $E_1 = \{(b'_i, w'_i) \mid 1 \le i \le q_W\} \cup \{(b'_{i+q_W}, h'_{2,i}) \mid 1 \le i \le q_H\}$;
      $E' = E_1$;
    else
      Let $E_1 = \{(b'_i, w'_i) \mid 1 \le i \le q_W\} \cup \{(b'_{i+q_W}, h'_{2,i}) \mid 1 \le i \le q_B - q_W\}$;
      Let $E_2 = \{(h'_{1,2i-1+q_W-q_B}, h'_{2,2i+q_W-q_B}) \mid 1 \le i \le \lfloor (q_H + q_W - q_B)/2 \rfloor\}$;
      if $(q_H + q_W - q_B)$ is odd **then**
        Let $E_2 = E_2 \cup (h'_{1,q_H-1}, h'_{2,q_H})$;
      end if
      $E' = E_1 \cup E_2$;
    end if
  end if
  return E';
end procedure

Proof. By Lemma 11. □

4.2.3 Case 2.2.3: G' is null

Let q_B, q_W, and q_H be the numbers of isolated black, white, and hybrid blocks, respectively. Without loss of generality, we assume that $q_B \ge q_W$. Our algorithm is shown in Algorithm 10.

Theorem 21 $|\mathrm{LOW}_{f2e}(\mathrm{BB}(G))| = |\mathrm{LOW}_{f2e}(\mathrm{BB}(\mathrm{BB}(G) \cup E_{added}))| + |E_{added}|$, *where* E_{added} *is the set of added edges returned by Algorithm 10.*

Proof. Let $|B'| = q_B$, $|W'| = q_W$, and $|H'| = q_H$. It is straightforward to see that the number of edges added by Algorithm 10 is $\max\{q_B, \lceil (q_B + q_W + q_H)/2 \rceil\}$. After adding edges, the number of isolated vertices is reduced to zero such that $|B| = q_B$. If $q_B > q_W + q_H$, then $|W| + |H| = q_W + q_H - 1$; otherwise, $|W| + |H| = q_W + q_H - [(q_W + q_H - q_B) \bmod 2]$. By Lemma 18, $|\mathrm{LOW}_{f2e}(\mathrm{BB}(\mathrm{BB}(G) \cup E_{added}))| = \max\{|B|, \lceil (|B| + |W| + |H|)/2 \rceil\}$. Therefore, the theorem holds. □

4.3 Complexity analysis

Theorem 22 *Algorithm 1 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 1 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 $\mathcal{M}$ PRAM to compute BB(G). After computing BB(G), the algorithm takes $O(1)$ time to determine which case should be executed. We now show that all cases in our algorithm take $O(1)$ time to add edges.

Case 1.1.1: It is trivial to show that Algorithm 2, i.e., ETCT, runs in $O(1)$ time.

Case 1.1.2: Similarly, Algorithm 3, i.e., AETC, runs in $O(1)$ time to add edges.

Case 1.1.3: The method runs in $O(1)$ time, since there are only five ways to add edges in this case.

Case 1.2.1: In this case, it takes $O(1)$ time to add a set of edges E_1 to the subgraph T' of BB(G), where the subgraph T' has an equal number of black and white leaves. Let $T'' = \text{BB}(G) - T'$ and $T' \cup E_1$ be a 2-edge-connected graph. Then, it takes $O(1)$ time to add edges between T" and BB($T' \cup E_1$) Therefore, Algorithm 4, i.e., BGTWAug, obviously runs in $O(1)$ time.

Case 1.2.2: Algorithm 5, i.e., HAssign, runs in $O(1)$ time to recolor the selected leaves and Algorithm 4, i.e., BGTWAug, also runs in $O(1)$ time. Therefore, this case can be solved in $O(1)$ time.

Case 2: As all sub-cases of Case 2 can be reduced to Case 1, it is trivial to show that these reduction methods run in $O(1)$ time. □

5 Componentwise 2-edge-connectivity augmentation

In this section, we present Algorithm 11, i.e., C2Aug, which solves the componentwise 2-edge-connectivity augmentation problem.

If the leaves in a graph are not all black, i.e., there is a white or a hybrid leaf in the graph, we can use the minimum number of added edges to make all tree edges non-bridge edges.

Lemma 23 *Algorithm 11 is correct and optimal.*

Algorithm 11 Componentwise 2-edge-connectivity augmentation

```
 1: procedure C2Aug(G)
 2:     Let T = BB(G);
 3:     Let S be the set of isolated vertices in T;
 4:     if there are leaves in T then
 5:         if the leaves are not all black then
 6:             T' = T − S;
 7:         else  {* Without loss of generality, assume that all leaves are black. *}
 8:             if there is an isolated vertex v in S whose corresponding block contains a white vertex then
 9:                 Let T' = T − S ∪ {v};
10:             else
11:                 T' = T − S;
12:             end if
13:         end if
14:         E=FS2Aug(T');  {* Algorithm 1 *}
15:     else
16:         E = ∅;
17:     end if
18:     return E;
19: end procedure
```

Proof. We divide the proof into two cases:

Case 1: If BB(G) contains at least one white leaf or one hybrid leaf.

In this case, all the black leaves can be connected to a white leaf, or a hybrid leaf if necessary. Since no bridges are adjacent to any isolated vertices, they can be removed from BB(G) without affecting the correctness of our algorithm. Let T' be the forest obtained from BB(G) after removing all of the isolated vertices. It is straightforward to see that the set of edges found by Algorithm 11 is equal to the set of edges found by Algorithm 1 with input T'. Therefore, our algorithm is correct and optimal.

Case 2: If all the leaves in BB(G) are black.

In this case, we remove all isolated vertices from BB(G), except the one whose corresponding block contains a white vertex. Let the resulting graph be T'. It is straightforward to see that the number of added edges is equal to the number of black leaves, which is equal to $\text{LOW}_{f2e}(T')$. Therefore, the lemma holds.

□

Theorem 24 *Algorithm 11 runs in sequential linear time and $O(\log n)$ parallel time on an EREW PRAM using a linear number of processors.*

Proof. By Fact 1, it 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 $\mathcal{M}$ PRAM to compute BB(G). Obviously, BB(G) can be reduced to a subgraph as input for Algorithm 1 in $O(1)$ time. Since, by Theorem 22, Algorithm 1 runs in sequential linear time and $O(\log n + \mathcal{T}_{\mathcal{M}}(n,m))$ parallel time, Algorithm 11 also runs in sequential linear time and $O(\log n + \mathcal{T}_{\mathcal{M}}(n,m))$ parallel time. □

6 Concluding remarks

We have considered two augmentation problems related to bipartite graphs. The first is a fundamental graph-theoretical problem. The second focuses on how to suppress the smallest amount of sensitive information in a cross-tabulated table, so that the resulting table does not leak important or confidential information. The latter is a fundamental issue concerning the security of statistical data. In both cases, after adding edges, the resulting graph is simpler than the input graph and does not contain any bridges. It can be either a simple graph or, if necessary, a multi-graph. The proposed approach determines whether or not such an augmentation is feasible.

We have described algorithms for finding a smallest 2-edge-connectivity augmentation and a smallest componentwise 2-edge-connectivity augmentation of input graphs. If there are two black and two white vertices in the input graph, the resulting graph will be simple. The algorithms can be trivially parallelized to run in optimal $O(\log n)$ time using a linear number of EREW processors.

References

[1] N. R. Adam and J. C. Wortmann. Security-control methods for statistical database: A comparative study. *ACM Computing Surveys*, 21:515–556, 1989.

[2] F. Y. Chin and G. Özsoyoğlu. Auditing and inference control in statistical databases. *IEEE Transactions Software Engineering*, 8:574–582, 1982.

[3] 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.

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

[5] L. H. Cox. Suppression methodology and statistical disclosure control. *Journal of the American Statistical Association*, 75:377–385, 1980.

[6] D. E. Denning and J. Schlörer. Inference controls for statistical databases. *IEEE Computer*, 16:69–82, July 1983.

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

[8] 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.

[9] D. Gusfield. A graph theoretic approach to statistical data security. *SIAM Journal on Computing*, 17:552–571, 1988.

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

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

[12] T.-s. Hsu. Undirected vertex-connectivity structure and smallest four-vertex-connectivity augmentation (extended abstract). In J. Staples, editor, *Lecture Notes in Computer Science 1004: Proceedings of the 6th International Symposium on Algorithms and Computation*, pages 274–283. Springer-Verlag, New York, NY, 1995.

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

[14] T.-s. Hsu. Simpler and faster vertex-connectivity augmentation algorithms (extended abstract). In M. Paterson, editor, *Lecture Notes in Computer Science 1879: Proceedings of the 8th European Symposium on Algorithms*, pages 278–289, New York, NY, 2000. Springer-Verlag.

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

[16] T.-s. Hsu and M. Y. Kao. Security problems for statistical databases with general cell suppressions. In *Proceedings of the 9th International Conference on Scientific and Statistical Database Management*, pages 155–164, 1997.

[17] 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.

[18] T.-s. Hsu and V. Ramachandran. On finding a smallest augmentation to biconnect a graph. *SIAM Journal on Computing*, 22:889–912, 1993.

[19] 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.

[20] G. Kant. *Algorithms for Drawing Planar Graphs*. PhD thesis, Utrecht University, the Netherlands, 1993.

[21] M. Y. Kao. Linear-time optimal augmentation for componentwise bipartite-completeness of graphs. *Information Processing Letters*, pages 59–63, 1995.

[22] M. Y. Kao. Data security equals graph connectivity. *SIAM Journal on Discrete Mathematics*, 9:87–100, 1996.

[23] M. Y. Kao. Total protection of analytic-invariant information in cross-tabulated tables. *SIAM Journal on Computing*, 26:231–242, 1997.

[24] J. P. Kelly, B. L. Golden, and A. A. Assad. Cell suppression: Disclosure protection for sensitive tabular data. *Networks*, 22:397–417, 1992.

[25] F. M. Malvestuto and M. Moscarini. Censoring statistical tables to protect sensitive information: Easy and hard problems. In *Proceedings of the 8th International Conference on Scientific and Statistical Database management*, pages 12–21, 1996.

[26] F. M. Malvestuto and M. Moscarini. Suppressing marginal totals from a two-dimensional table to protect sensitive information. *Statistics and Computing*, 7:101–114, 1997.

[27] F. M. Malvestuto, M. Moscarini, and M. Rafanelli. Suppressing marginal cells to protect sensitive information in a two-dimensional statistical table. In *Proceedings of the 10th ACM SIGACT-SIGMOD-SIGACT Symposium on Principles of Database Systems*, pages 252–258, 1991.

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

[29] 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.

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

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

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

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