OPTIMAL AUGMENTATION FOR BIPARTITE COMPONENTWISE BICONNECTIVITY IN LINEAR TIME

TSAN-SHENG HSU*AND MING-YANG KAO†

Abstract. A graph is componentwise fully biconnected if every connected component either is an isolated vertex or is biconnected. We consider the problem of adding the smallest number of edges to make a bipartite graph componentwise fully biconnected while preserving its bipartiteness. This problem has important applications for protecting sensitive information in cross tabulated tables. This paper presents a linear-time algorithm for the problem.

Key words. algorithm, linear time, bipartite graph augmentation, biconnectivity, data security

AMS(MOS) subject classifications. 68Q20, 68R10, 94C15, 05C40, 05C90

1. Introduction. The problem of adding the minimum number of edges to make a given graph biconnected is called the *smallest biconnectivity augmentation problem.* This problem has been extensively studied for general graphs [4, 17, 18]. A simplified sequential algorithm which corrects an error in [18] and an efficient parallel algorithm are reported in [13]. Efficient algorithms for other vertex connectivity augmentation problems can be found in [9, 10, 20].

A graph is *componentwise fully biconnected* if every connected component either is biconnected or is an isolated vertex. This paper presents a linear-time algorithm for the problem of adding the smallest number of edges into a given bipartite graph to make it componentwise fully biconnected while maintaining its bipartiteness. Our problem arises naturally from research on data security of cross-tabulated tables [16]. To protect sensitive information in a cross tabulated table, it is a common practice to suppress some of the cells in the table. A fundamental issue concerning the effectiveness of this practice is how a table maker can suppress a small number of cells in addition to the sensitive ones so that the resulting table does not leak significant information. This protection problem can be reduced to augmentation problems for bipartite graphs [6, 7, 11, 14, 16, 15]. In particular, a linear-time algorithm for our augmentation problem yields a linear-time algorithm for suppressing the minimum number of additional cells so that no nontrivial information about any individual row or column is revealed to an adversary [16].

In §2, we formally define our problem and give some basic definitions. In §3, we review a data structure for representing a graph that is not biconnected. In §4, we give a lower bound on the minimum number of additional edges necessary for achieving the desired bipartite biconnectivity. We prove a matching upper bound for a special case in §5 and for the general case in §6. We conclude this paper with a linear-time algorithm for our augmentation problem in §7.

* Institute of Information Science, Academia Sinica, Nankang 11529, Taipei, Taiwan, ROC. Research supported in part by NSC 84-2213-E-001-005.

† Department of Computer Science, Duke University, Durham, North Carolina 27706, USA. Research supported in part by NSF Grant CCR-9101385.

2. Problem formulation and basic definitions.

In this paper, all graphs are undirected and have neither self loops nor multiple edges, unless explicitly stated otherwise. Given a graph G', an edge subset E' and a vertex subset V', $G - V'$ denotes G' without the vertices in V' and their adjacent edges. $G' - E'$ is G' without the edges in E'. $G' \cup E'$ is the resulting G' after the edges in E' are added to G'. Most of the definitions below can be found in [2, 8, 12, 13].

Throughout this paper, let $G = (A, B, E)$ denote a bipartite graph.

2.1. The augmentation problem.

A *trivial* connected component is an isolated vertex. A *cut vertex* (or *cut edge*) is one whose removal increases the number of connected components. A connected graph is *biconnected* if it has at least three vertices and no cut vertex. A *biconnected component* is a maximal subgraph that is biconnected. A graph is *fully biconnected* if it either consists of a single vertex or is biconnected. A graph is *componentwise fully biconnected* if every connected component is fully biconnected.

A *legal edge* of G is an edge in $A \times B$ but not in E. A *biconnector* of G is a set L of legal edges such that $G \cup L$ is componentwise fully biconnected. A biconnector is *optimal* if it is one with the smallest number of edges. Note that if $A = \emptyset$ or $B = \emptyset$, G is trivially componentwise fully biconnected. If $|A| = 1$ and $B \neq \emptyset$ (or $|B| = 1$ and $A \neq \emptyset$), G has no biconnector. If $|A| \geq 2$ and $|B| \geq 2$, G has a biconnector. In light of these observations, the *optimal biconnector problem* is the following: given $G = (A, B, E)$ with $|A| \geq 2$ and $|B| \geq 2$, find an optimal biconnector of G.

The remainder of this paper assumes that $|A| \geq 2$ and $|B| \geq 2$.

2.2. Basics.

A *block* in a graph is either the set of a single vertex that is not in any biconnected component or the set of vertices in a biconnected component. A block with exactly one vertex is a *singular block*. Let $nc(G)$ denote the number of connected components in G. A *strict* cut vertex is a cut vertex c such that (1) c is not an endpoint of a cut edge, or (2) $nc(G - \{c\}) - nc(G) \geq 2$. A singular block consisted of a strict cut vertex is a *strict cut block*.

DEFINITION 2.1 ([12, 13]). A block is a *leaf-block* if it either (1) is a singular block which contains one endpoint of a cut edge or (2) contains exactly one strict cut vertex and no endpoint of any cut edge. A vertex is *demanding* if (1) it is the only vertex in a leaf-block or (2) it is neither a cut vertex nor an endpoint of a cut edge.

We classify the vertices and leaf-blocks of G with the following definitions. A vertex is of *type-A* if it is in A. A vertex is of *type-B* if it is in B. A leaf-block is of *type-A* if all of its demanding vertices are in A. A leaf-block is of *type-B* if all of its demanding vertices are in B. A block is of *type-AB* if it has at least one demanding vertex in A and one demanding vertex in B.

LEMMA 2.2.

1. *A biconnected component in a bipartite graph must contain at least two vertices in A and at least two vertices in B.*
2. *A nontrivial leaf-block is type-AB.*
3. *A singular leaf-block is either type-A or type-B.*

Proof. Straightforward. □

Let Ω' be a set of leaf-blocks in G. A *legal pair* of Ω' is two distinct elements in Ω' that are paired according to the following rules. Type-A may pair with type-B or type-AB. Type-B may pair with type-A or type-AB. Type-AB may pair with all three types.

A *corresponding* legal edge of G for a legal pair is a legal edge whose two endpoints are demanding vertices in the blocks. A *legal matching* of Ω' is a set of legal pairs of Ω' such that each element in Ω' is contained in at most one legal pair. A legal matching of Ω' with the largest cardinality can be obtained by iteratively applying any rule below whenever applicable:

- If all unpaired elements are type-AB, we pair two type-AB elements.
- If there are one unpaired type-A element and one unpaired type-B element, we pair a type-A element and a type-B element.
- If there is no unpaired type-B element and there are one unpaired type-A element and one unpaired type-AB element, we pair a type-A element with a type-AB element.
- If there is no unpaired type-A element and there are one unpaired type-B element and one unpaired type-AB element, we pair a type-B element with a type-AB element.

$\Omega(G)$ denotes the set of leaf-blocks of G. For $\Omega' \subseteq \Omega(G)$, $\mathcal{M}(\Omega')$ denotes the maximum cardinality of a legal matching of Ω'. For a maximum legal matching of Ω', $\mathcal{R}(\Omega')$ denotes the number of elements in Ω' that is not in the given maximum legal matching. Note that $\mathcal{R}(\Omega')$ is the same for any maximum legal matching of Ω'.

For all vertices $u \in G$, $\mathcal{D}(u, G)$ denotes the number of connected components in $X - \{u\}$ where X is the connected component of G containing u. $\mathcal{C}(G)$ denotes the number of connected components in G that are not fully biconnected. $\mathcal{B}(G)$ denotes the number of edges in an optimal biconnector of G. The next notation denotes our target size for an optimal biconnector of G:

$$\alpha(G) = \max_{u \in G}\{\mathcal{D}(u, G) + \mathcal{C}(G) - 2, \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))\}.$$

Note that $\alpha(G) = O(n)$, where n is the number of vertices in G.

3. A bc-forest.

We construct a forest $\Psi(G)$, called the *bc-forest* of G, to organize the non-strict cut blocks, cut edges, and strict cut vertices in G. Our augmentation algorithm works with this forest instead of G directly. The construction below is a variant of the bc-forest given in [8, 19, 12]. Let $Y_1, \ldots, Y_b$ be the blocks of G that are not strict cut blocks. Let $u_1, \ldots, u_c$ be the strict cut vertices. Let $e_1, \ldots, e_w$ be the cut edges.

The vertex set of $\Psi(G)$ is $\{Y_1, \ldots, Y_b\} \cup \{u_1, \ldots, u_c\} \cup \{e_1, \ldots, e_w\}$, i.e., each non-strict cut block, strict cut vertex or isolated edge of G is regarded as a vertex in $\Psi(G)$. The vertices in $\Psi(G)$ corresponding to blocks are called the *b-vertices*, and those corresponding to strict cut vertices and cut edges are called the *c-vertices*.

The edge set of $\Psi(G)$ is the union of the sets $\{(Y_i, e_j) \mid$ an endpoint v of e_j is in Y_i and v is not a strict cut vertex$\}$, $\{(u_i, e_j) \mid u_i$ is an endpoint of $e_j\}$, and $\{(Y_i, u_j) \mid u_j \in Y_i\}$. In other words, there is an edge between Y_i and u_j if and only if u_j is a vertex in the block Y_i. There is an edge between Y_i and e_j if and only if one endpoint of e_j is in the block Y_i and this endpoint is not a strict cut vertex. There is an edge between u_i and e_j if and only if u_i is an endpoint of e_j. Remark that if a strict cut vertex v forms a block $\{v\}$ by itself, then $\{v\}$ does not appear in the bc-forest as a b-vertex. Instead, v appears as a c-vertex in the bc-forest.

By our definition of a bc-forest, given any path P between two vertices u and v, P has a c-vertex if u and v are both b-vertices. If u and v are both c-vertices with degree at least three, then P has a b-vertex. It is also true that a b-vertex that is no isolated is adjacent to only c-vertices. The number of vertices in $\Psi(G)$ is $O(n)$, where n is the number of vertices in G.

LEMMA 3.1 ([8]).

1. *$\Psi(G)$ is a forest, where the leaves are leaf-blocks in G.*
2. *Each connected component in G forms a tree in $\Psi(G)$. Moreover, a connected component that is trivial or biconnected forms an isolated vertex in $\Psi(G)$.*
3. *A leaf-block of G is a b-vertex of degree one in $\Psi(G)$.*
4. *For all cut vertices u in G, $\mathcal{D}(u, G)$ equals the degree of u in $\Psi(G)$.*

It is obvious to update $\Psi(G)$ after adding a legal edge to G between demanding vertices in two distinct connected components. We simply add a new cut edge to G to connect the two trees corresponding the two connected components in question. The next lemma analyzes the case of adding an edge within a connected component.

LEMMA 3.2 ([4, 8, 18]). *For simplicity, assume that G is connected. Let Y_1 and Y_2 be leaf-blocks in G. Let e be a legal edge between a demanding vertex in Y_1 and one in Y_2. Let $G' = G \cup \{e\}$. Let P be the tree path of $\Psi(G)$ between Y_1 and Y_2.*

1. *The blocks of $\Psi(G)$ on P are merged into a new block Y' in $\Psi(G')$. All the other blocks remain the same.*
2. *The c-vertices of $\Psi(G)$ on P that are of degree two are no longer c-vertices in $\Psi(G')$. All the other c-vertices remain the same.*
3. *If a c-vertex is adjacent to a block on P in $\Psi(G)$, then it is adjacent to the new block Y' in $\Psi(G')$. All the other edges remain the same.*

4. A lower bound on the size of optimal biconnectors.

LEMMA 4.1. *G is componentwise fully biconnected if and only if $\alpha(G) = 0$.*

Proof. Straightforward. □

The next lemma is useful for bounding the size of an optimal biconnector.

LEMMA 4.2. *G has an optimal biconnector L such that the connected components of G which are not fully biconnected are all contained in the same connected component of $G \cup L$.*

Proof. Let K be an optimal biconnector of G. If K connects all connected components of G which are not fully biconnected, then the lemma is true. Otherwise, let X_1 and X_2 be two connected components of G which are not fully biconnected and are contained in two different connected components X'_1 and X'_2 of $G \cup K$, respectively. Let $e_1 = (u_1, v_1)$ and $e_2 = (u_2, v_2)$ be two edges in K with $e_1 \in X'_1$ and $e_2 \in X'_2$. Such e_1 and e_2 exist because X_1 and X_2 are not fully biconnected in G, but X'_1 and X'_2 are fully biconnected in $G \cup K$. Next, let $e'_1 = (u_1, v_2)$ and $e'_2 = (u_2, v_1)$. Then, $K' = (K \setminus \{e_1, e_2\}) \cup \{e'_1, e'_2\}$ is an optimal biconnector of G. Also, K' connects $X'_1 - \{e_1\}$ and $X'_2 - \{e_2\}$, which include X_1 and X_2.

The lemma is proven by repeating this endpoint switching process. ☐

A nontrivial connected component is *biconnectable* if it is neither an isolated edge nor a biconnected component. To motivate for the lower bound on $\mathcal{B}(G)$, observe that if G is made componentwise fully biconnected, then G has neither isolated edges nor biconnectable components. In light of Lemma 4.2, legal edges should be added to G between demanding vertices in biconnectable components and isolated edges to merge all isolated edges and biconnectable components into one biconnectable component. By Lemmas 3.2, legal edges should be added between demanding vertices to componentwise fully biconnect the resulting biconnectable component. Note that if G is componentwise fully biconnected, then $\Psi(G)$ contains no leaf. Adding a legal edge to G can merge at most two leaf-blocks into a block that might or might not be a leaf-block. These observations suggest the next theorem.

THEOREM 4.3. *$\mathcal{B}(G) \geq \alpha(G)$.*

Proof. Note that $\alpha(G) = \max_{u \in G}\{\mathcal{D}(u,G) + \mathcal{C}(G) - 2, \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))\}$. We first prove that $\mathcal{B}(G) \geq \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))$. If G is componentwise fully biconnected, then $|\Omega(G)| = \mathcal{M}(\Omega(G)) = 0$. Thus, it suffices to prove that adding a legal edge can decrease $\mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))$ by at most one. If the two endpoints in the added legal edge are in two leaf-blocks that are a legal pair, then $\mathcal{M}(\Omega(G))$ decreases by at most one and $\mathcal{R}(\Omega(G))$ remains unchanged. Otherwise, either $\mathcal{R}(\Omega(G))$ decreases by at most one and $\mathcal{M}(\Omega(G))$ remains unchanged or $\mathcal{M}(\Omega(G))$ decreases by one and $\mathcal{R}(\Omega(G))$ increases by one. We now prove that $\mathcal{B}(G) \geq \max_{u \in G}\{\mathcal{D}(u,G) + \mathcal{C}(G) - 2\}$. If G is componentwise fully biconnected, then $\mathcal{D}(u,G) \leq 1$, $\mathcal{C}(G) = 0$, and the lemma holds. Otherwise, let $X_1, \ldots, X_{\mathcal{C}(G)}$ be the connected components in G that are not fully biconnected. Let v be a vertex such that $\mathcal{D}(v,G) + \mathcal{C}(G) - 2 = \max_{u \in G}\{\mathcal{D}(u,G) + \mathcal{C}(G) - 2\}$. Then, v must be a cut vertex. Without loss of generality, assume that $v \in X_1$. By Lemma 4.2, G has an optimal biconnector L such that the connected components X_i are all contained in some connected component X' in $G \cup L$. Because the removal of v breaks X_1 into $\mathcal{D}(v,G)$ connected components, $X' - \{v\}$ contains $\mathcal{D}(v,G) + \mathcal{C}(G) - 1$ connected components of $G - \{v\}$. Because X' is biconnected, $X' - \{v\}$ is connected and L must contain at least $\mathcal{D}(v,G) + \mathcal{C}(G) - 2$ edges. ☐

5. Determining the optimal biconnector size for a special case.

In this section, we consider the case when the graph is connected with with at least two vertices in A and two vertices in B. During the discussion, we also assume that $|\Omega(G)| > 3$, since otherwise it is obvious to prove $\alpha(G) = \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))$. Note that $\alpha(G) = \max_{u \in G}\{\mathcal{D}(u,G) - 1, \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))\}$ for this case. The next theorem is the main result of this section.

THEOREM 5.1. *There is a legal edge e such that $\alpha(G \cup \{e\}) = \alpha(G) - 1$.*

Before we prove this theorem, we give the next corollary.

COROLLARY 5.2. *$\mathcal{B}(G) = \alpha(G)$.*

Proof. Theorem 5.1 can be iteratively applied to add $\alpha(G)$ edges to G so that the $\alpha(\cdot)$ value of the resulting G is zero. By Lemma 4.1, this resulting G is componentwise fully biconnected. Thus, $\mathcal{B}(G) \leq \alpha(G)$. The corollary then follows from Theorem 4.3. ☐

We will prove Theorem 5.1 through a sequence of lemmas in §5.1–§5.4.

DEFINITION 5.3 ([13]).

1. A cut vertex u is *massive* if $\mathcal{D}(u,G) - 1 > \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))$.

2. A cut vertex u is *critical* if $\mathcal{D}(u, G) - 1 = \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))$.
3. A graph with no massive c-vertex is *balanced.*

LEMMA 5.4 ([4, 13, 18]). *Assume that $\Psi(G)$ has more than three leaves.*

1. There is at most one massive vertex.
2. If there is a massive vertex, then there is no critical vertex.
3. There are at most two critical vertices.
4. If there are two critical vertices, then $|\Omega(G)| = 2 \cdot \mathcal{M}(\Omega(G))$ and $\mathcal{R}(\Omega(G)) = 0$.

LEMMA 5.5. *At least one of the following four cases holds for G.*

- *Case I. $\mathcal{M}(\Omega(G)) = 0$.*
- *Case II. $\mathcal{M}(\Omega(G)) > 0$ and there are two critical c-vertices.*
- *Case III. $\mathcal{M}(\Omega(G)) > 0$, G is balanced and there is at most one critical c-vertex.*
- *Case IV. $\mathcal{M}(\Omega(G)) > 0$ and there is one massive c-vertex.*

Proof. By definition, either Case I holds or $\mathcal{M}(\Omega(G)) > 0$. The latter case is further divided by Lemma 5.4. □

In §5.1 through §5.4, the proof of Theorem 5.1 is divided into the four cases stated in Lemma 5.5.

5.1. Case I of Theorem 5.1. This case assumes $\mathcal{M}(\Omega(G)) = 0$.

LEMMA 5.6. $\alpha(G) = |\Omega(G)|$.

Proof. Let u be a vertex in G. If u is a noncut vertex, by definition $\mathcal{D}(u, G) = 1$; otherwise, by Lemma 3.1(4), $\mathcal{D}(u, G) \leq |\Omega(G)|$. Then, because G is connected (i.e., $\mathcal{C}(G) = 1$) and $\mathcal{M}(\Omega(G)) = 0$, $\alpha(G) = |\Omega(G)|$. □

LEMMA 5.7. *Theorem 5.1 holds for Case I.*

Proof. Let $k = |\Omega(G)|$. By Theorem 4.3 and Lemma 5.6, it suffices to construct a biconnector L of k edges for G. Let $Y_1, \ldots, Y_k$ be the leaf-blocks in G. Because $\mathcal{M}(\Omega(G)) = 0$, by the way a legal matching is defined these blocks are all type-A or all type-B. By Lemma 2.2(2), each block is also singular. Assume without loss of generality that they are type-A. Then, there is a cut edge (x_i, y_i) for each $Y_i = \{y_i\}$. Because we assume there are at least two type-B vertices in G and $\mathcal{M}(\Omega(G)) = 0$, there are at least two distinct vertices u and v among $x_1, \ldots, x_k$. We construct an optimal biconnector L as follows:

- Let Ω_u (respectively, $\overline{\Omega}_u$) be the set of all Y_i with $x_i = u$ (respectively, $x_i \neq u$).
- For each $Y_i \in \Omega_u$ (respectively, $\overline{\Omega}_u$), let $e_i = (y_i, v)$ (respectively, (y_i, u)).
- Let L_u (respectively, $\overline{L}_u$) be the set of all e_i associated with the blocks in Ω_u (respectively, $\overline{\Omega}_u$).
- Let $L = L_u \cup \overline{L}_u$.

Thus $|L| = k$. To shows that L is an optimal biconnector, it remains to show that L is a biconnector of G. By Lemma 5.4(1), $\Psi(G)$ is a tree. By Lemma 3.1(3), the blocks in $\overline{\Omega}_u$ are leaves in $\Psi(G)$. We will prove that $G \cup L$ has no cut edge or cut vertex.

First we observe that adding edges to G creates neither a new cut edge nor a new cut vertex. For each cut edge e in G, $G - \{e\}$ consists of two connected components G_1 and G_2, where G_1 contains a vertex v_1 in Ω_u and G_2 contains a vertex v_2 in $\overline{\Omega}_u$. Note that there are two internally vertex-disjoint paths between v_1 and v_2 in $G \cup L$, one being v_1, v, v_2 and the other being v_2, u, v_1. Thus v_1 and v_2 are connected in $(G \cup L) - \{e\}$. Thus e is no longer a cut edge in $G \cup L$.

For each cut vertex c in G, $G - \{c\}$ consists of connected components $G_1, G_2, \ldots, G_r$. For each connected component G_i containing a vertex in Ω_u, we can find another connected component G_j containing a vertex in $\overline{\Omega}_u$. By an argument similar to the one given in the previous paragraph, G_i and G_j are connected in $(G \cup L) - \{c\}$. Thus $G_1, G_2, \ldots, G_r$ are all connected in $(G \cup L) - \{c\}$. This implies c is no longer a cut vertex in $G \cup L$. Thus, L is a biconnector of G. □

5.2. Case II of Theorem 5.1. This case assumes that $\mathcal{M}(\Omega(G)) > 0$ and there are exactly two critical vertices. Let $P = u_1, \ldots, u_k$ be a path in G. P is *branchless* if for all i with $1 < i < k$ the degree of u_i in G is two. P is *branching* if it is not branchless. A subtree rooted at a child of the root of a rooted tree is a *branch* of the given tree.

In this case, $\Psi(G)$ has a very special shape as described in the next lemma.

LEMMA 5.8 ([18]). *Let T be a tree with k leaves such that there are two vertices u and v each of degree $1 + \frac{k}{2}$.*

1. *The value k is even.*
2. *There is a unique branchless path in T that connects u and v.*
3. *For each leaf, there is a unique branchless path in T that connects the leaf to either u or v and does not go through the other.*

We call the T in Lemma 5.8 a *double star* centered at u_1 and u_2. The two vertices u_1 and u_2 are critical vertices. We say that a leaf is *clung* to u_i if it is connected to u_i through a path as described in the lemma.

By Lemma 5.4, $\mathcal{R}(\Omega(G)) = 0$ in this case. The next lemma analyzes how to construct a maximum legal matching of cardinality $\mathcal{M}(\Omega(G))$.

LEMMA 5.9. *There is a maximum legal matching of cardinality $\mathcal{M}(\Omega(G))$ such that each pair in it consists of a leaf-block clung to u_1 and one clung to u_2.*

Proof. We construct $\mathcal{M}(\Omega(G))$ as follows. Let $b_1, \ldots, b_{r_1}, b_{r_1+1}, \ldots, b_{r_2}, b_{r_2+1}, \ldots, b_{r_3}$ be the leaves clung to u_1, where b_i, $1 \le i \le r_1$ are type-A, b_i, $r_1 < i \le r_2$ are type-AB, and b_i, $r_2 < i \le r_3$ are type-B. Let $w_1, \ldots, w_{s_1}, w_{s_1+1}, \ldots, w_{s_2}, w_{s_2+1}, \ldots, w_{s_3}$ be the leaves clung to u_2, where w_i, $1 \le i \le s_1$ are type-B, w_i, $s_1 < i \le s_2$ are type-AB, and w_i, $s_2 < i \le s_3$ are type-B. By Lemma 5.8, $r_3 = s_3 = \ell$, where $2 \cdot \ell$ is the number of leaves in $\Omega(G)$. We construct a legal matching $Q = \{(b_i, w_i) \mid 1 \le i \le \ell\}$. In order for Q to be a legal matching, we must verify that $r_1 \le s_2$ and $r_2 \ge s_1$ in order for the pairs to satisfy the matching rule. Assume that $r_1 > s_2$, then there are $r_1 + (s_3 - s_2 + 1) \ge \ell$ type-A leaves. Hence $\mathcal{R}(\Omega(G)) > 0$. This is a contradiction to the assumption. Thus $r_1 \le s_2$. Using an argument that is similar to the above, we can prove $r_2 \ge s_1$. Since Q is a legal matching and $|Q| = \ell$, Q is a maximum legal matching. □

LEMMA 5.10. *Theorem 5.1 holds for Case II.*

Proof. We use Lemma 5.9 to find a maximum legal matching. For each pair in the matching, we find a legal pair. By adding all legal pairs, G is biconnected [4, 18]. The size of the maximal legal matching is equal to $\mathcal{M}(\Omega(G))$. □

5.3. Case III of Theorem 5.1. This case assumes that $\mathcal{M}(\Omega(G)) > 0$, there is no massive c-vertex, and there is at most one critical c-vertex.

Let c_1 be a c-vertex in $\Psi(G)$ of the largest degree among all c-vertices. The next lemma identifies the conditions under which adding an edge can decrease the number of leaves in $\Psi(G)$ by two.

DEFINITION 5.11 ([13]). Let P be the path in $\Psi(G)$ between two leaf-blocks B_1 and B_2 that are a legal pair. The pair B_1 and B_2 (or the path P) satisfy the *leaf-connecting condition* if P has (1) a b-vertex of degree at least four or (2) two vertices each of degree at least three.

LEMMA 5.12 ([13]). *Let G' be the resulting graph obtained from G after adding a corresponding legal edge between a legal pair that satisfy the leaf-connecting condition. Then $\Omega(G') = \Omega(G) - 2$ and $\mathcal{M}(\Omega(G')) + \mathcal{R}(\Omega(G')) = \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G)) - 1$.*

LEMMA 5.13. *Assume that $\Psi(G)$ has more than three leaves and is rooted at a vertex r of degree more than two. Let h be a vertex in $\Psi(G)$ other than r with degree at least three. If $\mathcal{M}(\Omega(G)) > 0$, then there are two leaves w_1 and w_2 with the following properties:*

- *w_1 and w_2 are a legal pair;*
- *the path in $\Psi(G)$ between w_1 and w_2 passes through h and z, where z is a vertex of degree at least three and $z \neq h$.*

Proof. Let T_h be the subtree of $\Psi(G)$ rooted at h. There are three cases.

Case 1. T_h has a type-AB leaf. Let this leaf be w_1. Let w_2 be a leaf in a branch of r not containing h. Thus w_1 and w_2 are the pair we want, i.e., z is r.

Case 2. T_h has both type-A and type-B leaves, but no type-AB leaves. Let w_2 be a leaf in a branch of r not containing h. If w_2 is type-AB or type-A (respectively, type-B), then let w_1 be a type-B (respectively, type-A) leaf in T_h. Thus w_1 and w_2 are the pair we want, i.e., z is r.

Case 3. T_h has type-A (respectively, type-B) leaves only. Let w_1 be a leaf in T. Without loss of generality, w_1 is type-A. Since $\mathcal{M}(\Omega(G)) > 0$, some leaf w_2 can match with w_1. It is either the case that w_2 is in a branch of r not containing h or the case that there is a vertex $x \neq r$ in the path from h to r such that w_2 is a descendent of x. In the former case, let $z = r$. In the latter case, x is of degree at least three. Thus let $z = x$. □

LEMMA 5.14. *Some legal pair in $\Psi(G)$ satisfy the leaf-connecting condition. Moreover, if c_1 is critical, the path in $\Psi(G)$ between this legal pair also contains c_1.*

Proof. There are two cases.

Case 1. The degree of c_1 is two. Since $|\Omega(G)| > 3$, there is no critical vertex. We root $\Psi(G)$ at a b-vertex r of the largest degree. Thus it is either the case that the degree of r is at least four or the case that the degree of r is three and there is another vertex r' of degree at least three. In the former case, using a simple argument we can prove that there must exist two distinct leaves w_1 and w_2 such that they are in different branches of r and they are a legal pair. This pair satisfy Condition (1) of the leaf-connecting condition. In the latter case, we use Lemma 5.13 by setting $h = r'$ to find a legal pair that satisfy Condition (2) of the leaf-connecting condition.

Case 2. The degree of c_1 is greater than two. Since c_1 is not massive, there must exist another vertex r of degree at least three. We root $\Psi(G)$ at r and then use Lemma 5.13 by setting $h = c_1$ to find a legal pair that satisfy Condition (2) of the leaf-connecting condition. □

LEMMA 5.15. *Theorem 5.1 holds for Case III.*

Proof. By Lemma 5.14, $\mathcal{D}(c_1, G') = \mathcal{D}(c_1, G) - 1$ and $\mathcal{M}(\Omega(G')) + \mathcal{R}(\Omega(G')) = \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G)) - 1$. □

5.4. Case IV of Theorem 5.1. This case assumes that $\mathcal{M}(\Omega(G)) > 0$ and there is exactly one massive c-vertex.

Let r be the massive cut vertex of G with $\mathcal{D}(r, G) - 1 > \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))$. For technical convenience, consider $\Psi(G)$ as a tree rooted at r.

LEMMA 5.16. $\alpha(G) = \mathcal{D}(r, G) - 1 > \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G)) > \mathcal{D}(u, G)$ *for all cut vertices u in G such that $u \neq r$.*

Proof. The lemma follows from the assumptions that there is exactly one massive c-vertex and there is no critical vertex. □

Based on Lemma 5.16, to prove Case IV of Theorem 5.1, we will add a legal edge to G to reduce $\mathcal{D}(r, G)$. Note that adding a legal edge to G never increases $\mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))$ or $\mathcal{D}(u, G)$.

LEMMA 5.17. $\mathcal{D}(r, G) \geq 4$.

Proof. Straightforward. □

A branch of T is a *chain* if it contains exactly one leaf in T.

LEMMA 5.18. *Let $\Psi(G)$ be rooted at a non-leaf vertex. Then $\Psi(G)$ contains two leaves Y_1 and Y_2, and two distinct branches T_1 and T_2 with the following properties:*

1. *T_1 is a chain.*
2. *Y_1 is in T_1 and Y_2 is in T_2.*
3. *Y_1 and Y_2 form a legal pair.*

Proof. By Lemmas 3.1(4) and 5.4, the number of children of r in $\Psi(G)$ is greater than half the number of leaves in $\Psi(G)$. Therefore, at least one branch of r is a chain. Let T_1 be such a chain. Let Y_1 be the unique leaf of T_1. Because $\mathcal{M}(\Omega(G)) > 0$, $\Psi(G)$ contains a leaf that is different from Y_1 and forms a legal pair with Y_1. Let Y_2 be such a leaf. Let T_2 be the branch of r that contains Y_2 as a leaf. By choice, Y_1, Y_2, T_1 and T_2 satisfy the three desired properties. □

LEMMA 5.19. *Theorem 5.1 holds for Case IV.*

Proof. Let Y_1, Y_2, T_1 and T_2 be two leaves and two subtrees of $\Psi(G)$ that satisfy Lemma 5.18. Let e be a legal edge between a demanding vertex in Y_1 and one in Y_2. Let $G' = G \cup \{e\}$. Let P be the tree path of $\Psi(G)$ between Y_1 and Y_2. By Lemma 3.2(1), the blocks of G on P are merged into a new block Y' in G'. By Lemmas 3.2(2) and 5.17, r remains a cut vertex in G'. Because T_1 has only one leaf, by Lemma 3.2(3), it is absorbed into T_2 in $\Psi(G')$. Thus, $\mathcal{D}(r, G') = \mathcal{D}(r, G) - 1$. By Lemma 3.2(3), $\mathcal{D}(v, G') \leq \mathcal{D}(v, G)$ for all remaining cut vertices v of G in G'. Because $\mathcal{D}(r, G)$ is greater than $\mathcal{D}(u, G)$ for all other cut vertices u in G, $\mathcal{D}(r', G)$ is at least $\mathcal{D}(v, G')$ for all other cut vertices v in G'. On the other hand, if T_2 has exactly one leaf, then Y' is a leaf in $\Psi(G')$; otherwise, it is an inner vertex. In either case, $\mathcal{M}(\Omega(G')) + \mathcal{R}(\Omega(G')) \leq \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))$. Thus, $\alpha(G') = \alpha(G) - 1$. We repeat this process until the massive cut vertex becomes critical, which is Case III of Theorem 5.1. □

6. A tight bound for the general case.

Let $\mathcal{C}_1(G)$ be the number of connected components in G that are not biconnected and have two or more vertices each. Let $\mathcal{C}_2(G)$, $\mathcal{C}_3(G)$ and $\mathcal{C}_4(G)$ be the numbers of isolated edges, isolated vertices, and biconnected components, respectively.

THEOREM 6.1.

1. *If* $\mathcal{C}_1(G) = 1$ *and* $\mathcal{C}_2(G) = 0$, *then* $\mathcal{B}(G) = \alpha(G)$.
2. *If* $\mathcal{C}_1(G) + \mathcal{C}_2(G) \geq 2$, *then* $\mathcal{B}(G) = \alpha(G)$.
3. *If* $\mathcal{C}_1(G) = 0$, $\mathcal{C}_2(G){=}1$, *and* $\mathcal{C}_4(G){\geq}1$, *then* $\mathcal{B}(G) = 2$.
4. *If* $\mathcal{C}_1(G) = 0$, $\mathcal{C}_2(G){=}1$, *and* $\mathcal{C}_4(G){=}0$, *then* $\mathcal{B}(G) = 3$.
5. *If* $\mathcal{C}_1(G) = 0$ *and* $\mathcal{C}_2(G){=}0$, *then* $\mathcal{B}(G) = 0$.

Proof. Case 5 is obvious. The other four cases are proved below.

Case 1. Let G_1 be the nontrivial connected component in G that is not biconnected. Theorem 5.1 covers the case that G_1 contains at least two vertices in A and at least two in B. Thus, we assume without loss of generality that G_1 contains exactly one vertex $u \in A$. Hence G_1 is a star centered at u. Let the other vertices in G_1 be $v_1, v_2, \ldots, v_r$. Then every $\{v_i\}$ is a leaf-block and $\mathcal{B}(G) = \Omega(G_1)$. If there is an isolated vertex $w \in B$, let $L = \{(w, v_1), \ldots, (w, v_r)\}$ and $G \cup L$ is biconnected. If there is a biconnected component G_1 in G, by Lemma 2.2 G_1 contains two vertices $w_1, w_2 \in B$ and $\{(w_1, v_1)\} \cup \{(w_2, v_2), (w_2, v_3), \ldots, (w_2, v_r)\}$ is an optimal biconnector.

Case 2. Case 1 proves that if $\mathcal{C}_1(G) = 1$ and $\mathcal{C}_2(G) = 0$, then $\mathcal{B}(G) = \alpha(G)$. We show by the following iterative algorithm that by adding one edge at a time carefully, we can reduce this case to Case 1. We add an edge by one of the following two subcases depending on the current value of $\mathcal{M}(\Omega(G))$.

Case 2.1. $\mathcal{M}(\Omega(G)) > 0$. We can find a legal pair w_1 and w_2. If they are in different connected components, then we add a corresponding legal edge. Let G' be the resulting graph. Then $\mathcal{C}_1(G') + \mathcal{C}_2(G') = \mathcal{C}_1(G) + \mathcal{C}_2(G) - 1$. Since $\mathcal{C}(G') = \mathcal{C}(G) - 1$ and $\mathcal{M}(\Omega(G')) = \mathcal{M}(\Omega(G)) - 1$, $\max_{u \in G}\{\mathcal{D}(u, G') + \mathcal{C}(G') - 2, \mathcal{M}(\Omega(G')) + \mathcal{R}(\Omega(G'))\} = \max_{u \in G}\{\mathcal{D}(u, G) + \mathcal{C}(G) - 2, \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))\} - 1$. If w_1 and w_2 are in the same connected component, then there is a vertex w_3 in another connected component such that w_1 and w_3 are also a legal pair. We then apply an argument similar to that for the case that w_1 and w_2 are in different connected components.

Case 2.2. $\mathcal{M}(\Omega(G)) = 0$. Thus $\mathcal{C}_2(G) = 0$. Without loss of generality, all leaves are type-A. We can find a vertex x in a type-A leaf-block and a type-B vertex y in another connected component. Let G' be the resulting graph after adding the edge (x, y). Then $\mathcal{C}(G') = \mathcal{C}(G) - 1$ and $\mathcal{R}(\Omega(G')) = \mathcal{R}(\Omega(G)) - 1$. Thus $\max_{u \in G}\{\mathcal{D}(u, G') + \mathcal{C}(G') - 2, \mathcal{M}(\Omega(G')) + \mathcal{R}(\Omega(G'))\} = \max_{u \in G}\{\mathcal{D}(u, G) + \mathcal{C}(G) - 2, \mathcal{M}(\Omega(G)) + \mathcal{R}(\Omega(G))\} - 1$. In both cases, we combine two connected components into a connected component with more than two vertices. Thus it is impossible that $\mathcal{C}_1(G) = 0$ and $\mathcal{C}_2(G) = 1$.

Note that by adding edges one by one at a time according to the above steps. We will eventually reach the point that $\mathcal{C}_1(G) = 1$ and $\mathcal{C}_2(G) = 0$ in the current G. Thus we can apply Case 1 to wrap up the proof.

Case 3. Let G' be a biconnected component in G. Let r and c be two vertices in G', where r is type-A and c is type-B. Let (r', c') be the isolated edge of G. We choose $L =$

$\{(r, c'), (r', c)\}$, which is obviously an optimal biconnector of G.

Case 4. Let (r, c) be the isolated edge and let r' and c' be two isolated vertices such that r and r' are type-A, and c and c' are type-B. We choose $L = \{(r, c'), (r', c), (r', c')\}$, which is clearly an optimal biconnector of G. □

7. Computing an optimal biconnector in linear time.

Since $\Psi(G)$ can be computed in linear time [1, 3, 5], so can $\alpha(G)$. To find an optimal biconnector, we can iteratively add one legal edge at a time to reduce $\alpha(G)$ by one. With a naive implementation, this process may take quadratic time to find an optimal biconnector. However, with the data structure presented below, we can find an optimal biconnector in linear time. It is obvious to compute in linear time an optimal biconnector for Cases 3 through 5 of Theorem 6.1. For Cases 1 and 2, if $\mathcal{C}_1(G) > 1$, these two cases can be reduced in linear time to the case that $\mathcal{C}_1(G) = 1$ and $\mathcal{C}_i(G) = 0$ for $2 \leq i \leq 4$. It is also easy to compute an optimal biconnector in linear time for Cases I, II and IV of Theorem 5.1. For the rest of the section, we assume Case III of Theorem 5.1.

Given two blocks in $\Psi(G)$, their *corresponding path* is the tree path in $\Psi(G)$ that contains those two blocks. The linear time algorithm in [18] uses the fact that $\Psi(G)$ is rooted at an internal b-vertex. Each time an edge e is added, it is added between leaf-blocks whose corresponding path P passes through the root. Thus $\Psi(G \cup e)$ can be computed from $\Psi(G)$ by local operations in $O(|P|)$ time. Note that if P passes the root, then the new root of $\Psi(G \cup e)$ is the new block created by merging all blocks in P.

Unfortunately, the key step of our algorithm as given in the proof of Lemma 5.14 (which uses Lemma 5.13) cannot guarantee that P passes through the root because we have to find specified leaf-blocks satisfying the matching rules in addition to satisfying the leaf-connecting condition. We will prove in the following sections that we can satisfy the requirement of P passing the root, if we reroot $\Psi(G)$ while finding the legal edge to be added. In order to have a linear-time implementation, the total amount of time used for rerooting and finding two endpoints of the legal edge to be added must be also linear. Below, we describe a data structure and also a new proof for Lemma 5.14 to achieve all of the above goals.

7.1. Data structure.

A vertex u in $\Psi(G)$ is *pure-A* if it is a type-A leaf or all leaves in the subtree rooted at u are type-A. We similarly define a *pure-B* vertex. A vertex u in $\Psi(G)$ is *hybrid* if it is neither type-A nor type-B. Note that a branch of a hybrid vertex may or may not contain a type-AB leaf.

We use the following data structure to represent $\Psi(G)$ as a tree rooted at a given non-leaf b-vertex. The siblings of a vertex are doubly linked from left to right. In this list, hybrid vertices whose branches contain type-AB leaves appear first. Hybrid vertices whose branches contain no type-AB leaves appear next. These vertices are followed by the pure-A vertices and then the pure-B vertices. Each vertex has two values: (1) a flag indicating whether it is pure-A, pure-B, or hybrid and (2) the number of leaves in the subtree rooted at it. Each hybrid vertex also keeps the number of type-AB leaves in the subtree rooted at it. Each vertex maintains five pointers: (1) a parent pointer, (2) a child pointer to the leftmost pure-A child, (3) a child pointer to the leftmost pure-B child, (4) a pointer to the leftmost hybrid child who has a branch containing a type-AB leaf and (5) a pointer to the leftmost hybrid child whose branches contain no type-AB leaf. Each pointer is null if no such vertex exists. In addition to the above pointers, each vertex also has a pointer to the leftmost child and one to the rightmost child.

Given $\Psi(G)$, we can construct the above data structure and root $\Psi(G)$ at a given non-leaf b-vertex all in linear time. This construction and rooting process is called *ordering*. The resulting $\Psi(G)$ together the data structure is called an *ordered tree*.

Using the ordered tree data structure, we can *walk down* the tree from a vertex v toward one of its child pointers. If there is a leaf with a certain type (i.e., type-A, -B, or -AB) in a subtree rooted at a vertex v. Using one of the 5 child pointers, we can walk down the tree to find such a leaf with the given type. We can *walk up* the tree from the vertex v through its parent pointer. We can *walk down* the tree from a vertex v toward one of its dependents u by first find the path P between u and v by walking up through the parent pointer until v is encountered. Then we follow P to walk down the tree. All of the walking operations mentioned above take time linear in the distance (number of vertices and edges) traversed.

LEMMA 7.1. *Let e be a legal edge in G whose endpoints are in leaf-blocks w_1 and w_2. If the path P between w_1 and w_2 in the ordered tree $\Psi(G)$ passes through the root of the tree, then we can order $\Psi(G \cup \{e\})$ in $O(|P|)$ time.*

Proof. By Lemma 3.2, all blocks in P are merged into the root. Let this newly created block be the new root. We can keep track of the flag and pointers of each vertex, except its parent pointer, all in $O(|P|)$ time. Since a parent pointer of a vertex points to its parent as before in $\Psi(G)$ or to the root of the resulting ordered tree, we can update the parent pointers of the vertices in P only. The order of each sibling link can be properly updated in $O(|P|)$ time. By keeping track of the current root, we can order $\Psi(G \cup \{e\})$ in $O(|P|)$ time. □

7.2. Rerooting and path finding.

LEMMA 7.2. *Let T be an ordered tree rooted at r. Let r' be another vertex in T. Let P be the path in T from r to r'. Given P, we can order T to be rooted at r' in $O(|P|)$ time.*

Proof. The proof is straightforward. □

LEMMA 7.3. *The vertices w_1, w_2, and z of Lemma 5.13 can be found in $O(|P|)$ time in an ordered $\Psi(G)$, where P is the tree path between w_1 and w_2.*

Proof. We consider the same three cases as in the proof of Lemma 5.13.

Case 1. This case is straightforward.

Case 2. We can find w_2 by first walking up $\Psi(G)$ from h until we reach the root. Then we walk down $\Psi(G)$ from a branch different from the one we walk up. Thus w_2 can be any leaf in this branch. A type-A (respectively, type-B) leaf w_1 can be found by walking down $\Psi(G)$ from h and chooses the leftmost (respectively, rightmost) child pointer each time we walk down.

Case 3. We can find w_1 by walking down $\Psi(G)$ from h using an arbitrary child pointer. We walk up $\Psi(G)$ from h until we reach the root. Either we find z on the way up or $z = r$. The rest of the proof is straightforward. □

Note that in Case 3, rerooting $\Psi(G)$ at z is needed if the root is a b-vertex and we require the corresponding path between the two found leaves to pass the root. Note also that rerooting happens only when the current root is pure-A or pure-B.

7.3. Choosing a legal pair in Lemma 5.14. We now give another proof for Lemma 5.14 on an ordered $\Psi(G)$, which may have to be rerooted. The rerooting process will be performed only if necessary. For ease of description, we split Lemma 5.14 into two lemmas which corresponds to the two cases in its proof.

LEMMA 7.4. *Lemma 5.14 holds for an ordered $\Psi(G)$ if each c-vertex is of degree two.*

Proof. Let r be the current root of the ordered tree. The vertex r is a b-vertex. Note that the degree of every c-vertex is two. Since $\Psi(G)$ has at least four leaves, there is no critical vertex. There are three cases.

Case 1. The root r is of degree at least four. The leftmost and the rightmost leaves are the desired legal pair. This pair satisfy Condition (1) of the leaf-connecting condition.

Case 2. The root r is of degree three. Let h be the root of a branch of r that is not a chain. Since $\Psi(G)$ has at least four leaves, there exists a vertex $h \neq r$ whose degree is at least three. We use Lemmas 5.13 and 7.3 to find the desired legal pair.

Case 3. The root r is of degree two. If both branches of r are not chains, then the leftmost and the rightmost leaves are a legal pair. This pair satisfy Condition (2) of the leaf-connecting condition. If one branch of r is a chain, then let r' be the first vertex of degree at least three when we walk down $\Psi(G)$ from r through the branch of r that is not a chain. Since the degree of every c-vertex is two, the vertex r' is a b-vertex. We reroot $\Psi(G)$ at r'. We reroot $\Psi(G)$ at r'. We can now reduce this case to either Case 1 or Case 2 depending on the degree of the new root r'. □

Let T be the original rooted tree. Let T' be the rerooted tree obtained in Case 3. Note that the rerooting operation performed in Case 3 moves the root from r to r', where every vertex in the corresponding path between r and r' is of degree 2. Let $T_{r'}$ be the subtree of T rooted at r'. Let T'_r be the subtree of T' rooted at r. It is true that whenever this rerooting operation is performed, $T_{r'}$ contains more than one leaf and T'_r is a chain.

LEMMA 7.5. *Lemma 5.14 holds for an ordered $\Psi(G)$ if the degree of some c-vertex is greater than two.*

Proof. Let c_1 be a c-vertex of the largest degree among all c-vertices. Let r be the current root of $\Psi(G)$. If there is a critical vertex in $\Psi(G)$, then c_1 is that vertex. The vertex r is a b-vertex. Let T_1 be the branch of r containing c_1. Let $P_{x,y}$ be the corresponding path in T between two vertices x and y. There are three cases.

Case 1. The degree of r is at least three. If c_1 is a hybrid vertex, we use Lemmas 5.13 and 7.3 by setting $h = c_1$ to find a legal pair. No rerooting is needed. Otherwise we have the following three subcases. Assume without loss of generality that c_1 is pure-A. Let y be the first non-pure-A vertex encountered on the way walking down $\Psi(G)$ from r to c_1. Since c_1 is pure-A, $y \in P_{r,c_1}$. The degree of y is at least three.

Case 1.1. There is a branch T_2 of r, $T_2 \neq T_1$, containing a leaf that is not type-A. We can find a leaf in T_2 and a leaf in T_1 to form a legal pair. This pair satisfy Condition (2) of the leaf-connecting condition.

Case 1.2. Every branch of r other than T_1 is pure-A and y is a b-vertex. If y is a b-vertex, we reroot $\Psi(G)$ at y and use Lemmas 5.13 and 7.3 by setting $h = y$ to find a legal pair.

Case 1.3. Every branch of r other than T_1 is pure-A and y is a c-vertex. If y is a c-vertex, we reroot $\Psi(G)$ at y' where y' is the first b-vertex encountered by walking down $\Psi(G)$ from y towards c_1. The vertex y' is in P_{y,c_1} because there is a b-vertex between two c-vertices whose degrees are more than two. We find an arbitrary leaf w_1 in the subtree rooted at c_1. We can find another leaf w_2 in the subtree rooted at y such that w_1 and w_2 are a legal pair. The leaves w_1 and w_2 satisfy Condition (2) of the leaf-connecting condition. The path between them passes through c_1 and r.

Case 2. The degree of r is two and the branch T_2 of r not containing c_1 is not a chain. Let T_{c_1} be the subtree rooted at c_1. We have three subcases.

Case 2.1. T_2 is not a chain and c_1 is hybrid. We can find a legal pair w_1 and w_2 where $w_1 \in T_{c_1}$ and $w_2 \in T_2$. This pair satisfy Condition (2) of the leaf-connecting condition.

Case 2.2. T_2 is not a chain, c_1 is pure (assume without loss of generality is pure-A), and T_2 is non-pure-A. This case is similar to Case 2.1.

Case 2.3. T_2 is not a chain, c_1 is pure (assume without loss of generality is pure-A), and T_2 is pure-A. Thus there is a non-pure-A vertex in P_{r,c_1} and the degree of this vertex is at least three. We reroot $\Psi(G)$ at y or y' where y and y' are defined in Case 1.2 and Case 1.3. We also apply the same method to find a legal pair as given in Case 1.2 or Case 1.3.

Case 3. The degree of r is two and the branch T_2 of r not containing c_1 is a chain. We first walk down $\Psi(G)$ from r to c_1 and find the first vertex v whose degree is at least three.

Case 3.1. $v \neq c_1$ and v is a b-vertex. We reroot $\Psi(G)$ at v. We can apply the method given in Case 1 to find a legal pair.

Case 3.2. $v \neq c_1$ and v is a c-vertex. We reroot $\Psi(G)$ at the first b-vertex encountered when walking down $\Psi(G)$ from v towards c_1. Such b-vertex exists since there is a b-vertex between the corresponding path of two c-vertices whose degrees are at least three. We can apply the method given in Case 2 to find a legal pair.

Case 3.3. $v = c_1$. Let w be a child of c_1 with where the subtree rooted at w is not a chain. Such a vertex w exists because c_1 is not massive. We reroot $\Psi(G)$ at w. We can apply the method given in Case 2 to find a legal pair.

□

COROLLARY 7.6. *The path between the legal pair found in Lemmas 7.4 and 7.5 passes through the root of* $\Psi(G)$ *after rerooting if necessary.*

Proof. Straightforward. □

We define a *rerooting operation* to be the process of moving the root (if needed) from its current root before applying the proof of Lemma 5.14 to find a legal pair to its new root after finding a legal pair. We define a *rerooting step* to be the rerooting of a tree from its current root r to a child w of r. We say the above rerooting step *begins* from r and *stops* at w. Given w, a rerooting step can be done in constant time. Note that to reroot a tree from its current root r to a new root r', we may consider this as the process of applying rerooting steps along the path between r and r'. In order to bound the total time of rerooting during the execution of finding all legal pairs, the following lemma bounds the number of rerooting steps.

LEMMA 7.7. *Let T be the input bc-forest. During the entire execution of our algorithm* *for finding all legal pairs, the rerooting steps are applied $O(q)$ times, where q is the number of vertices in T.*

Proof. Assume that we need to perform a rerooting from r to r'. Let the path between r and r' be $r = w_1, w_2, \ldots, w_{k-1}, w_k = r'$. Note that r and r' are both b-vertices. Thus w_{k-1} is a c-vertex. We decompose a rooting operation into a sequence of rooting steps w_1 to w_2, w_2 to $w_3, \ldots, w_{k-1}$ to w_k. Those rerooting steps are classified into two categories. The first category consists of those rerooting steps from w_s to w_{s+1}, $1 \leq s \leq k-3$. The second category consists of the rerooting steps from w_{k-2} to w_{k-1} and from w_{k-1} to w_k. We collect all rerooting operations performed during the entire execution of finding all legal pairs. Those rerooting operations are decomposed into rerooting steps. All rerooting steps are classified into the above two categories. We will analyze the number of rerooting steps in each category.

For a rooting step from w to w' in the first category, the following observations are useful. Let T be the rooted tree before the rerooting step and let T' be the rooted tree after the rerooting step. Either the vertex w is a pure vertex in T' or the subtree of T' rooted at w is a chain. Before the rerooting step the vertex w' is neither pure-A nor pure-B in T. The subtree rooted at r' in T is also not a chain. After the rerooting step, a pure-A (respectively, pure-B) vertex in T remains pure-A (respectively, pure-B) in T'. Given any vertex v, no rerooting step in the first category stops at v twice. Thus the total number of rerooting steps in the first category is $O(q)$, where q is the number of vertices in the bc-forest of the given graph.

Note that for each legal pair found, we apply the rerooting operation once. For each rerooting operation, there are two rerooting steps in the second category. Note that we found only $O(q)$ legal pairs. Thus the total number of rerooting steps in the second category is $O(q)$. □

THEOREM 7.8. *Given a bipartite graph with n vertices and m edges, an optimal biconnector can be computed in $O(n+m)$ time.*

Proof. It takes $O(n+m)$ time in total to first construct $\Psi(G)$ and then maintain it while adding edges [18]. By Lemma 7.1 and Corollary 7.6, each update of the ordered $\Psi(G)$ takes time linear in the length of the path between the chosen legal pair. Thus by Lemmas 7.3, 7.4, 7.5, and 7.7, using the data structure of [18] to maintain the degrees of c-vertices, it takes $O(|P|)$ time in total to find a legal pair each time we are in Case III of Theorem 5.1. It also takes $O(|P|)$ time to update $\Psi(G)$. After each update, the size of $\Psi(G)$ decreases by $|P|$. Eventually, $\Psi(G)$ is an isolated vertex. Since the size of $\Psi(G)$ is linear in the size of G, the total time spent in Case III of Theorem 5.1 is linear. We have already shown that Cases I, II, and IV can be processed in linear time. Hence we prove the theorem. □

Acknowledgments.. The authors wish to thank Dan Gusfield for helpful discussions. The second author is grateful to Dan for his constant encouragements.

REFERENCES

[1] A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass., 1974.

[2] B. Bollobás, *Graph Theory: An Introductory Course*, Springer-Verlag, New York, 1979.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[4] K. P. Eswaran and R. E. Tarjan, *Augmentation problems*, SIAM J. Comput., 5 (1976), pp. 653–665.

[5] S. Even, *Graph Algorithms*, Computer Science Press, Rockville, MD, 1979.

[6] D. Gusfield, *Optimal mixed graph augmentation*, SIAM J. Comput., 16 (1987), pp. 599–612.

[7] ——, *A graph theoretic approach to statistical data security*, SIAM J. Comput., 17 (1988), pp. 552–571.

[8] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.

[9] T.-s. Hsu, *On four-connecting a triconnected graph (extended abstract)*, in Proc. 33rd Annual IEEE Symp. on Foundations of Comp. Sci., October 1992, pp. 70–79.

[10] ——, *Undirected vertex-connectivity structure and smallest four-vertex-connectivity augmentation (extended abstract)*, in Proc. 6th International Symp. on Algorithms and Computation, vol. LNCS #1004, Springer-Verlag, 1995, pp. 274–283.

[11] T.-s. Hsu and M. Y. Kao, *Optimal bi-level augmentation for selectively enhancing graph connectivity with applications*, in Proc. 2nd International Symp. on Computing and Combinatorics, vol. LNCS #1090, Springer-Verlag, 1996, pp. 169–178.

[12] T.-s. Hsu and V. Ramachandran, *A linear time algorithm for triconnectivity augmentation*, in Proc. 32th Annual IEEE Symp. on Foundations of Comp. Sci., 1991, pp. 548–559.

[13] ——, *On finding a smallest augmentation to biconnect a graph*, SIAM J. Comput., 22 (1993), pp. 889–912.

[14] M. Y. Kao, *Total protection of analytic invariant information in cross tabulated tables*, in Lecture Notes in Computer Science 775: the 11th Symposium on Theoretical Aspects of Computer Science, 1994, pp. 723–734.

[15] ——, *Linear-time optimal augmentation for componentwise bipartite-completeness of graphs*, Information Processing Letters, (1995), pp. 59–63.

[16] ——, *Data security equals graph connectivity*, SIAM Journal on Discrete Mathematics, 9 (1996), pp. 87–100.

[17] J. Plesník, *Minimum block containing a given graph*, ARCHIV DER MATHEMATIK, XXVII (1976), pp. 668–672.

[18] A. Rosenthal and A. Goldner, *Smallest augmentations to biconnect a graph*, SIAM J. Comput., 6 (1977), pp. 55–66.

[19] W. T. Tutte, *Connectivity in Graphs*, University of Toronto Press, 1966.

[20] T. Watanabe and A. Nakamura, *A minimum 3-connectivity augmentation of a graph*, J. Comp. System Sci., 46 (1993), pp. 91–128.