In SIAM JOURNAL ON COMPUTING, 1993. Copyright SIAM

On Finding a Smallest Augmentation to Biconnect a Graph*

Tsan-sheng Hsu and *Vijaya Ramachandran*

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

March 31, 1992

Abstract. We consider the problem of finding a minimum number of edges whose addition biconnects an undirected graph. This problem has been studied by several other researchers, two of whom presented a linear time algorithm for this problem in an earlier volume of this journal. However that algorithm contains an error which we expose in this paper. We present a corrected linear time algorithm for this problem as well as a new efficient parallel algorithm. The parallel algorithm runs in $O(\log^2 n)$ time using a linear number of processors on an EREW PRAM, where n is the number of vertices in the input graph.

Key words. algorithm, linear time, graph augmentation, biconnected graph, parallel computation, poly-log time, EREW PRAM

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

*This work was supported in part by NSF Grant CCR-89-10707. This paper appears in *SIAM Journal on Computing*, 1993, pp. 889-912.

1 Introduction

The problem of augmenting a graph to reach a certain connectivity requirement by adding edges has important applications in network reliability [6, 12, 21] and fault-tolerant computing. One version of the augmentation problem is to augment the input graph to reach a given connectivity requirement by adding a smallest set of edges. We refer to this problem as the *smallest augmentation* problem.

The following results are known for solving the smallest augmentation problem on an undirected graph to satisfy a vertex connectivity requirement. Eswaran & Tarjan [4] gave a lower bound on the smallest number of edges for biconnectivity augmentation and proved that the lower bound can be achieved. Rosenthal & Goldner [18] developed a linear time sequential algorithm for finding a smallest augmentation to biconnect a graph. Watanabe & Nakamura [26, 28] gave an $O(n(n+m)^2)$ time sequential algorithm for finding a smallest augmentation to triconnect a graph with n vertices and m edges. Hsu & Ramachandran [11] developed a linear time algorithm for this problem. There is no polynomial time algorithm known for finding a smallest augmentation to k-vertex-connect a general graph, for $k > 3$. There is also no efficient parallel algorithm known to find a smallest augmentation to k-vertex-connect a graph for $k \geq 2$.

For the problem of finding a smallest augmentation for a graph to reach a given edge connectivity property, several polynomial time algorithms on undirected graphs, directed graphs and mixed graphs are known. These results can be found in Cai & Sun [1], Eswaran & Tarjan [4], Frank [5], Gusfield [8], Kajitani & Ueno [13], Naor, Gusfield & Martel [15], Ueno, Kajitani & Wada [24], Watanabe [25] and Watanabe & Nakamura [27]. Efficient parallel algorithms for finding smallest augmentations for 2-edge connectivity, strong connectivity and making a mixed graph strongly orientable can be found in Soroker [20].

Another version of the problem is to augment a graph, with a weight assigned to each edge, to meet a connectivity requirement using a set of edges with a minimum total cost. Several related problems have been proved to be **NP**-complete. These results can be found in Eswaran & Tarjan [4], Frank [5], Frederickson & Ja'Ja' [7], Watanabe & Nakamura [26] and Watanabe, Narita & Nakamura [29].

In this paper, we present an efficient parallel algorithm for finding a smallest augmentation to biconnect an undirected graph. In addition, we have discovered an error in the sequential algorithm of Rosenthal & Goldner [18]. We first give a corrected linear time sequential algorithm for the problem. Our efficient parallel algorithm is based on this corrected sequential algorithm. However we have to utilize several insights into the problem in order to derive the parallel algorithm. The algorithm runs in $O(\log^2 n)$ time using a linear number of processors on an EREW PRAM, where n is the number of vertices in the input graph. (For more on PRAM models and PRAM algorithms see Karp & Ramachandran [14].)

The algorithmic notation used is from Tarjan [22] and Ramachandran [17]. We enclose comments between '{*' and '*}'. We use the following **pfor** statement for executing a loop in parallel.

pfor *iterator* $\rightarrow$ *statement list* **rofp**

The effect of this statement is to perform the statement list in parallel for each value of the iterator. We use the following form for an **if** statement.

if $condition_1 \rightarrow statement\ list_1$
| $condition_2 \rightarrow statement\ list_2$
$\vdots$
| $condition_n \rightarrow statement\ list_n$
fi

The effect of this statement is to perform the first statement list whose corresponding condition is true. If there is no condition is true, none of the statement lists is evaluated. Parameters are called by value unless they are declared with the keyword **modifies** in which case they are called by value and result.

2 Definitions

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. Let $\{E_i | 1 \leq i \leq k\}$ be a partition of E into a set of k disjoint subsets such that two edges e_1 and e_2 are in the same partition if and only if there is a simple cycle in G containing e_1 and e_2 or e_1 is equal to e_2. A vertex is called an *isolated* vertex if it is not adjacent to any other vertex. Let q be the number of isolated vertices in G. Let $\{V_i | 1 \leq i \leq k + q\}$ be a collection of sets of vertices, where V_i is the set of vertices in E_i for each i, $1 \leq i \leq k$, and V_{i+k} contains only the ith isolated vertex for each i, $1 \leq i \leq q$. A vertex v is a *cutpoint* of a graph G if v appears in more than one vertex set V_i. G is *biconnected* if it has at least 3 vertices and contains no cutpoint or isolated vertex. The subgraph $G_i = (V_i, E_i)$, $\forall i$, $1 \leq i \leq k$, is a *biconnected* *component* of G if V_i contains more than two vertices. Note that $E_i = \emptyset$, $\forall i$, $k < i \leq k + q$, since V_i contains an isolated vertex. The subgraph $G_i = (V_i, E_i)$, $\forall i$, $1 \leq i \leq k + q$, is called a *block* of G. Given an undirected graph G, we can define its *block graph* $blk(G)$ as follows. Each block and each cutpoint of G is represented by a vertex of $blk(G)$. The vertices of $blk(G)$ which represent blocks are called *b-vertices* and those representing cutpoints are called *c-vertices*. Two vertices u and v of $blk(G)$ are adjacent if and only if u is a c-vertex, v is a b-vertex and the corresponding cutpoint of u is contained in the corresponding block of v or vice versa. It is well known that $blk(G)$ is a forest and if G is connected, $blk(G)$ is a tree. If $blk(G)$ is a tree, it is also called a *block tree*.

Let n_c be the number of c-vertices in $blk(G)$. A vertex v_i represents a c-vertex of $blk(G)$ and d_i is the degree of v_i. We assume that $d_i \geq d_{i+1}$, $\forall i$, $1 \leq i < n_c$ throughout the discussion. For convenience, we define $a_i = d_i - 1$. If $blk(G)$ is a tree, let T be the rooted tree obtained from $blk(G)$ by rooting $blk(G)$ at the b-vertex which connects to v_1 and is on the path from v_1 to v_2. We use T_i to represent the subtree of T rooted at v_i for each i, $1 \leq i \leq n_c$, and we use T' to represent the subtree of T after deleting T_1. Let l_i be the number of leaves of T_i, $\forall i$, $1 \leq i \leq n_c$. We also use T_v to represent the subtree rooted at a vertex v of $blk(G)$. The subgraph of T induced by deleting the vertex v is denoted by $T - v$.

In a forest, a vertex with degree 1 is a *leaf*. Let l be the number of leaves in $blk(G)$. For a graph G', we use l' to denote the number of degree-1 vertices in $blk(G')$. Let $d(v)$ be the degree of the vertex v in $blk(G)$ and let d be the largest degree of all c-**vertices** in $blk(G)$.

In figures, we use a rectangle to represent a b-vertex and a circle to represent a c-vertex. A line denotes an edge. A path in the block graph is represented by a thick dashed line while a polygon represents a collection of subtrees. These notations are shown in Figure 1.

We also need the following definitions. Part of Definition 4 is from [18].

Definition 1 *A vertex v of $blk(G)$ is called* massive *if and only if v is a c-vertex with $d(v) - 1 > \lceil \frac{l}{2} \rceil$. A vertex v of $blk(G)$ is* critical *if and only if v is a c-vertex with $d(v) - 1 = \lceil \frac{l}{2} \rceil$. The graph $blk(G)$ is* critical *if and only if there exists a critical c-vertex in $blk(G)$.*

Definition 2 *A block graph $blk(G)$ is* balanced *if and only if G is connected and without any massive c-vertex. (Note that $blk(G)$ could have a critical c-vertex.) A graph G is balanced if and only if $blk(G)$ is balanced.*

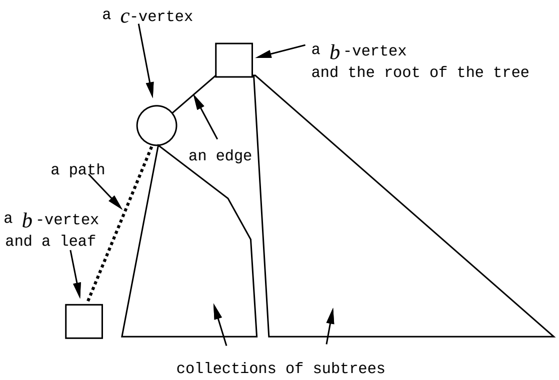

Figure 1: Notations for figures.

Definition 3 [The leaf-connecting condition]
Two leaves u_1 and u_2 of $blk(G)$ satisfy the leaf-connecting condition if and only if u_1 and u_2 are in the same tree of $blk(G)$ and the path P from u_1 to u_2 in $blk(G)$ contains either
(1) two vertices of degree more than 2,
or (2) one b-vertex of degree more than 3.

Definition 4 *Let v be a c-vertex of $blk(G)$. We call those components of $blk(G) - v$ which contain only one vertex of degree 1 in $blk(G)$* v-chains *[18]. A degree-1 vertex of $blk(G)$ in a v-chain is called a* v-chain leaf.

3 Main Lemmas

In this section, we present results that will be crucial in the development of our efficient parallel algorithm.

Lemma 1 *If $blk(G)$ has more than two c-vertices, then $a_1 + a_2 + a_3 - 1 \leq l$.*

Proof: Note that v_1 is a c-vertex with the largest degree. Vertex v_2 is a c-vertex with the largest degree among all c-vertices other than v_1. Vertex v_3 is a c-vertex with the largest degree among all c-vertices other than v_1 and v_2. Recall that if $blk(G)$ is a tree, we root $blk(G)$ at the b-vertex b which connects to v_1 and is on the path from v_1 to v_2. Let the rooted tree be T. Recall that T_i is the subtree of T rooted at v_i and l_i is the number of leaves in

T_i. T' is the subtree obtained from T by removing T_1. Let l_x be the number of leaves in T'. *Case 1:* If v_3 is in T_1, then $l_1 \geq a_1 - 1 + a_3$ and $l_x \geq a_2$. This implies $l = l_1 + l_x \geq a_1 + a_2 + a_3 - 1$. *Case 2:* If v_3 is in T', but not in T_2, then $l_1 \geq a_1$ and $l_x \geq a_2 + a_3$. Thus $l = l_1 + l_x \geq a_1 + a_2 + a_3$. *Case 3:* If v_3 is in T_2, then $l_1 \geq a_1$ and $l_x \geq l_2 \geq a_2 - 1 + a_3$. This implies $l = l_1 + l_x \geq a_1 + a_2 + a_3 - 1$.

Suppose that $blk(G)$ is a forest and v_1, v_2 and v_3 are in different trees T_1, T_2 and T_3, respectively. If v_i is the only c-vertex in T_i, then $a_i = l_i - 1$. Otherwise, $a_i \leq l_i$. Thus $a_1 + a_2 + a_3 \leq l_1 + l_2 + l_3 \leq l$. It is easy to prove the lemma for the case that $blk(G)$ is a forest and any two of v_1, v_2 and v_3 are in the same tree. $\square$

Corollary 1 *If $blk(G)$ has more than two c-vertices, then $a_3 \leq \frac{l+1}{3}$.*

<u>*Proof:*</u> From the definition, we know that $a_1 \geq a_2 \geq a_3$. If $a_3 > \frac{l+1}{3}$, then $a_1 \geq a_2 \geq a_3 > \frac{l+1}{3}$ which implies $a_1 + a_2 + a_3 > \frac{l+1}{3} * 3 = l + 1$. This is a contradiction to Lemma 1. $\square$

Corollary 2 *There can be at most one massive vertex in $blk(G)$.*

<u>*Proof:*</u> The corollary is obviously true if there are less than two c-vertices in $blk(G)$. If $blk(G)$ has only two c-vertices v_1 and v_2, there is a b-vertex b^* in $blk(G)$ that connects to both v_1 and v_2. We root $blk(G)$ at b^*. Since there are only two c-vertices, the children of v_1 and v_2 are all leaves. We know that a_1 and a_2 are equal to the number of children of v_1 and v_2 respectively, thus $a_1 + a_2 = l$. Suppose v_1 is massive, then $a_1 > \frac{l}{2}$. Thus $a_2 < \frac{l}{2}$. If $blk(G)$ has more than two c-vertices and v_1 and v_2 are massive, then $a_1 + a_2 > l$. Since $a_3 \geq 1$, we have derived a contradiction to Lemma 1. $\square$

Corollary 3 *If there is a massive vertex in $blk(G)$, then there is no critical vertex in $blk(G)$.*

<u>*Proof:*</u> The proof of Corollary 2 also applies here. $\square$

Corollary 4 *There can be at most two critical vertices in $blk(G)$, if $l > 2$.*

<u>*Proof:*</u> The corollary is obviously true if $blk(G)$ has only one or two c-vertices. Assume that $blk(G)$ has more than two c-vertices. From corollary 1, we know that $a_3 \leq \frac{l+1}{3}$. Since $\lceil \frac{l}{2} \rceil \geq \frac{l}{2} > \frac{l+1}{3}$, *if* $l > 2$, we know that v_3 cannot be critical if $l > 2$. $\square$

Before introducing the next lemma, we have to study properties for updating the block tree. The following fact for obtaining $blk(G')$ from $blk(G)$ is given in Rosenthal & Goldner [18].

Fact 1 *Given a graph G and its block tree $blk(G)$, adding an edge between two leaves u and v of $blk(G)$ creates a cycle C. Let G' be the graph obtained by adding an edge between u' and v' in G where u' and v' are non-cutpoint vertices in the blocks represented by u and v respectively. The following relations hold between $blk(G)$ and $blk(G')$.*
(1) Vertices and edges of $blk(G)$ that are not in the cycle C remain the same in $blk(G')$.
(2) All b-vertices in $blk(G)$ that are in the cycle C contract to a single b-vertex b' in $blk(G')$.
(3) Any c-vertex in C with degree equal to 2 is eliminated.
(4) A c-vertex x in C with degree greater than 2 remains in $blk(G')$ with edges incident on vertices not in the cycle. The vertex x also attaches to the b-vertex b' in $blk(G')$.

An example of forming $blk(G')$ from $blk(G)$ is illustrated in Figure 2.

Lemma 2 *Let u_1 and u_2 be two leaves of $blk(G)$ satisfying the leaf-connecting condition (Definition 3). Let α and β be non-cutpoint vertices in blocks of G represented by u_1 and u_2 respectively. Let G' be the graph obtained from G by adding an edge between α and β and let P represent the path between u_1 and u_2 in $blk(G)$. The following three conditions are true.*
(1) $l' = l - 2$.
(2) If v is a cutpoint in P with degree greater than 2 in $blk(G)$, then the degree of v decreases by 1 in $blk(G')$.
(3) If v is a cutpoint in P with degree equal to 2, then v is eliminated in $blk(G')$.

Proof: Parts (2) and (3) of the lemma follow from parts (3) and (4) of Fact 1. We now prove part (1) of the lemma.

From part (2) in Fact 1, we know that every vertex of G that is in a component represented by a b-vertex in P is in a biconnected component Q of G'. Let Q be represented by a b-vertex b in $blk(G')$.

Case 1: Suppose that part (1) of the leaf-connecting condition (Definition 3) holds. Let w and y be two vertices of $blk(G)$ having degree more than 2 in $blk(G)$ and let $blk(G')$ be rooted at b. In $blk(G)$, let w' be a vertex adjacent to w and y' be a vertex adjacent to y, with neither w' nor y' in P. The vertex b has at least two children, w' and y', in $blk(G')$ and

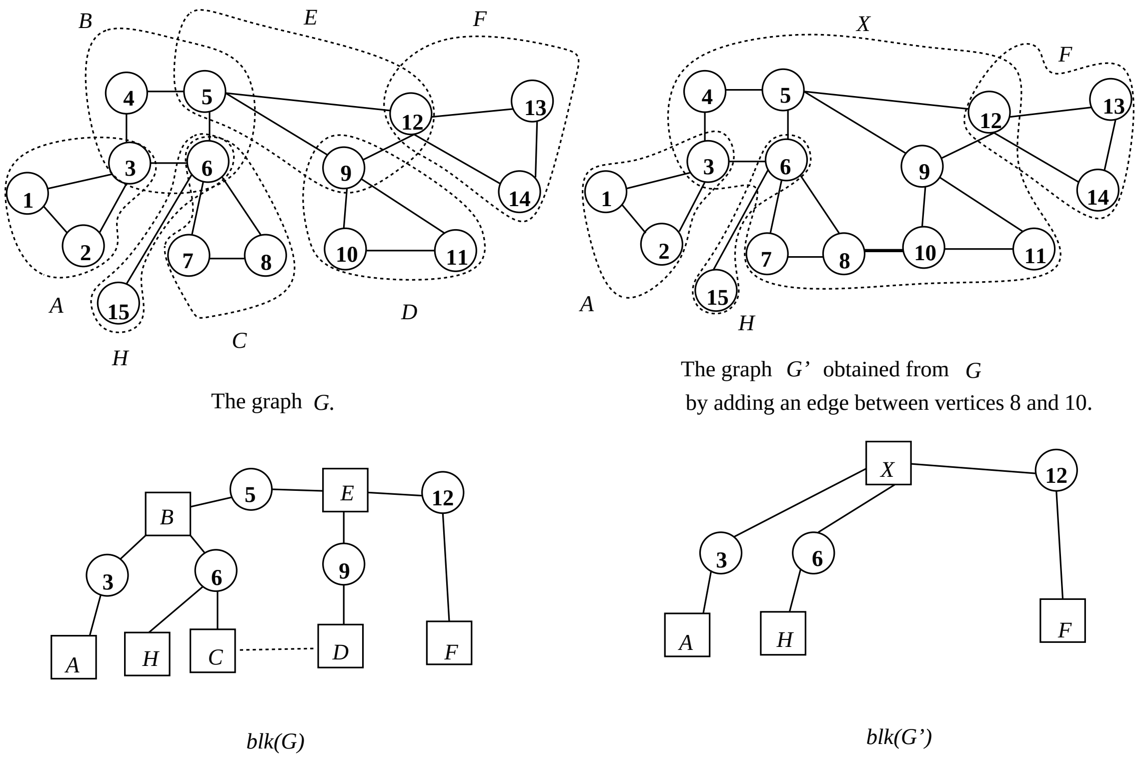

Figure 2: An example of obtaining $blk(G')$ from $blk(G)$. Vertices of G and G' circled with a dotted line are in the same block. For example, vertices 1, 2 and 3 of G are in block A. A vertex that appears in more than one block is a cutpoint. For example, vertex 3 appears in block A and B, thus it is a cutpoint. Vertices B, C, D and E in $blk(G)$ are in a cycle if we add an edge between C and D. The cycle contracts into a new b-vertex X in $blk(G')$. The degree of a c-vertex in the cycle decreases by 1 in $blk(G')$, if the original degree is more than two. A degree-2 c-vertex in the cycle is eliminated in $blk(G')$.

hence cannot be a leaf. Since leaves u_1 and u_2 are eliminated in $blk(G')$ and no new leaf is created, $l' = l - 2$.

Case 2: Suppose that part (2) of the leaf-connecting condition (Definition 3) holds. Let w be a b-vertex of degree more than 3. We can find at least two c-vertices, y' and z', connected to w, but not in P. The same reasoning used in case 1 can be followed to prove this case. □

4 The Algorithm

The original linear time sequential algorithm in Rosenthal & Goldner [18] consists of three stages. However, we have discovered an error in stage 3 of the algorithm in [18]. We present a corrected version of that stage of the algorithm in [18]. Our parallel algorithm follows the structure of the corrected sequential algorithm. The first two stages are easy to parallelize and we describe them in Section 4.1 and Section 4.2. However, stage 3 is highly sequential. Most of our discussion is on a corrected algorithm for stage 3 and its parallelization (Section 4.3).

We first state a lower bound on the number of edges needed to augment a graph to reach biconnectivity.

Theorem 1 *Eswaran & Tarjan [4]:* [Lower bound on the augmentation number]
Let G be an undirected graph with h connected components and let q be the number of isolated vertices in $blk(G)$. Then at least $max\{d + h - 2, \lceil \frac{l}{2} \rceil + q\}$ edges are needed to biconnect G, if $q + l > 1$.

4.1 Stage 1

Theorem 2 *Rosenthal & Goldner [18]*
Let G be an undirected graph with h connected components. We can connect G by adding $h-1$ edges, which we may choose to be incident on non-cutpoint vertices in blocks corresponding to leaves or isolated vertices in $blk(G)$.

Given $blk(G)$, stage 1 is easy to parallelize in time $O(\log n)$ optimally on an EREW PRAM by using the Euler tour technique described in Tarjan & Vishkin [23]. The block graph can be updated by creating a new b-vertex b and two new c-vertices c_v and c_w for each new edge (v,w). We create edges from b to c_v and b to c_w. Let b_v and b_w be the two b-vertices in the block graph whose corresponding blocks contain v and w, respectively. We create edges from c_v to b_v and from c_w to b_w.

4.2 Stage 2

Theorem 3 *Rosenthal & Goldner [18]*
Let G be connected and let v^* be a massive vertex in G. Let $\delta = d - 1 - \lceil \frac{l}{2} \rceil$. Then we can find at least $2\delta + 2$ v^*-chains. Let Q be the set of v-chain leaves. By adding $2k, k \leq \delta$ edges to connect $2k+1$ vertices of Q, we can reduce both the degree of the massive vertex and the number of leaves in the block tree by k.

Corollary 5 *Rosenthal & Goldner [18]*
Let G be connected and let v^* be a massive vertex in G. Let $\delta = d - 1 - \lceil \frac{l}{2} \rceil$ and let Q be the set of v^*-chain leaves. By adding 2δ edges to connect $2\delta + 1$ vertices of Q, we can obtain a balanced block tree.

In stage 2, v^*-chain leaves can be found by first finding the number of leaves in each subtree rooted at a child of v^*. A leaf is in a v^*-chain if and only if it is in a one-leaf subtree rooted at a child of v^*. Let Q be the set of vertices (excluding v^*) on cycles created by adding edges. The new block graph can be updated by merging vertices in Q into a single b-vertex b. Vertices b and v^* are connected by a new edge. These procedures can be done optimally in time $O(\log n)$ on an EREW PRAM.

4.3 Stage 3

In this stage, we have to deal with a graph G where $blk(G)$ is balanced. The idea is to add an edge between two leaves y and z under the conditions that the path P between y and z passes through all critical vertices and the new block tree has two less leaves if $blk(G)$ has more than 3 leaves. Thus the degree of any critical vertex decreases by 1 and the tree remains balanced.

In Rosenthal & Goldner [18], $blk(G)$ is rooted at a b-vertex b^*. A path P is found that contains two leaves y and z such that if $blk(G)$ contains two critical vertices v and w, P

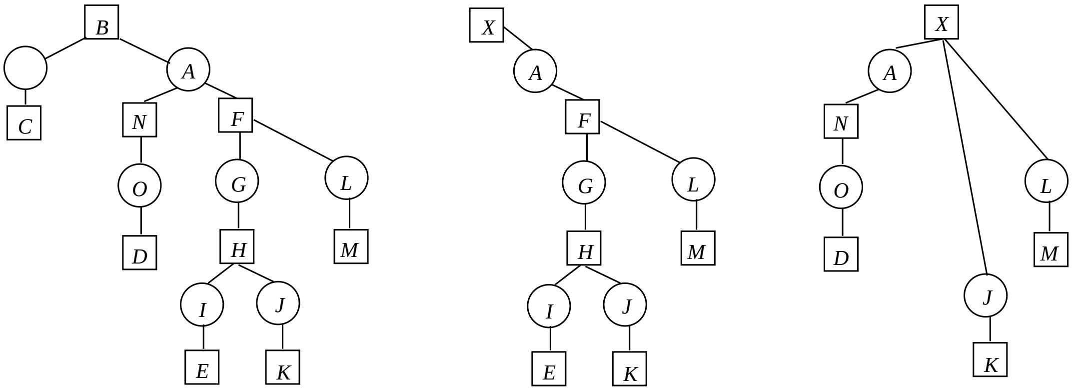

Figure 3: A counter example for the linear time sequential algorithm given by Rosenthal & Goldner. The left tree is $blk(G)$ rooted at B. Vertex A is the c-vertex with the largest degree. The middle tree is the new block tree after connecting two non-cutpoint vertices of G in the corresponding blocks represented by C and D. The number of leaves decreases by 1. The right tree is the new block tree after connecting two non-cutpoint vertices of G in the corresponding blocks represented by C and E. The number of leaves decreases by 2. The pair C and D could be chosen by the algorithm given by Rosenthal & Goldner while the pair C and E can be chosen to reduce the number of leaves by 2.

contains both of them. If $blk(G)$ contains less than 2 critical vertices, P contains b^* and a c-vertex with degree d (recall that d is the maximum degree of any c-vertex). It is possible that in the case that $blk(G)$ is balanced with more than three leaves and less than two critical vertices, P contains only one vertex of degree more than 2. If we add an edge between the two end points of P, it is possible that the new block tree has only one less leaf. An example of this is shown in Figure 3. Thus the lower bound cannot be achieved by this method.

We now give a corrected version of stage 3 which runs in linear time. Our method is based on the proof of the tight bound given in Eswaran & Tarjan [4], but we add an additional step to handle the case $d = 2$ (that is, $a_1 = 1$); the analysis of this case is omitted in [4]. We present our revised version of stage 3 below.

graph function seq_bca(**graph** G);
{$*$ G has at least 3 vertices and $blk(G)$ is balanced; l is the number of degree-1 vertices in $blk(G)$; $a_1 + 1$ is the largest degree of all c-vertices in $blk(G)$. $*$}
  tree T; **vertex** $v, w, y, z, \alpha, \beta$;
  let T be $blk(G)$ rooted at an arbitrary b-vertex;
  do $l \geq 2$ $\rightarrow$

if $a_1 = 1 \rightarrow$ **if** $l = 2 \rightarrow$ let v be any c-vertex in T; $w := v$
$\mid l > 2 \rightarrow$
1. let v be a b-vertex with degree greater than 2;{$*$Such a vertex must exist if $l > 2$ and $a_1 = 1$. $*$}
$w := v$;{$*$ This is the default value for w. $*$}
if $\exists$ a b-vertex in $T - v$ with degree greater than $2 \rightarrow$
2. let w be a b-vertex in $T - v$ with degree greater than 2
fi
fi
$\mid a_1 > 1 \rightarrow$
3. let v be a c-vertex with the largest degree in T;
if the largest degree for c-vertices in $T - v$ is greater than $2 \rightarrow$
4. let w be a c-vertex in $T - v$ with the largest degree
$\mid$ the largest degree for c-vertices in $T - v$ is less than 3
or there is no c-vertex in $T - v \rightarrow$
$w := v$;{$*$ This is the default value for w. $*$}
if $\exists$ a b-vertex in T with degree greater than $2 \rightarrow$
5. let w be a b-vertex in T with degree greater than 2
fi
fi
fi;
6. find two leaves y and z such that the path between them passes through v and w;
find a non-cutpoint vertex α in the corresponding block of G represented by y;
find a non-cutpoint vertex β in the corresponding block of G represented by z;
add an edge between α and β; update the block graph T
od;
return G
end seq_bca;

Claim 1 *If $blk(G)$ is balanced, we can biconnect G by adding $\lceil \frac{l}{2} \rceil$ edges using algorithm seq_bca.*

Proof: We first discuss the case when $blk(G)$ has more than 3 leaves. In this case, a critical vertex must have degree more than 2.

Case 1: If $blk(G)$ has two critical vertices v and w, then algorithm seq_bca finds them in steps 1 and 2 or 3 and 4, respectively.
Case 2: If $blk(G)$ has only one critical vertex v, algorithm seq_bca finds it in step 1 or step 3. Because $blk(G)$ is balanced and $l > 3$, there must exist another vertex w with degree more than 2. Otherwise v is massive. Algorithm seq_bca find w in one of the steps 1, 2, 4 or 5.
Case 3: The block tree $blk(G)$ has no critical vertex. Then either there is only one vertex (which must be a b-vertex) with degree more than 3 or there are two vertices with degrees more than 2. If there is only one vertex v with degree more than 3, algorithm seq_bca finds v in step 1. Suppose there are two vertices v and w with degrees more than 2 in the block tree. If v and w are both c-vertices, algorithm seq_bca finds them in steps 3 and 4, respectively. If v and w are both b-vertices, algorithm seq_bca finds them in steps 1 and 2, respectively. If one of v and w is a c-vertex and the other one is a b-vertex, algorithm seq_bca finds the c-vertex in step 3 and the b-vertex in step 5.

In all three cases, we can find two vertices of degree more than 2 or a b-vertex of degree more than 3. Thus by Lemma 2, the number of leaves in the new block tree reduces by two. Because v and w are the possible critical vertices, we reduce the value of d by 1. Thus the block tree remains balanced. Hence we can achieve the lower bound in Eswaran & Tarjan [4] by the algorithm.

For the case of $l = 3$, we can reduce $blk(G)$ into a new block tree with two leaves by picking any pair of leaves in $blk(G)$ and connecting them. We know that we can reduce a block tree of 2 leaves into a single vertex by connecting the two leaves. Thus the claim is true. $\square$

Claim 2 *Algorithm seq_bca runs in $O(n + m)$ time.*

Proof: The block tree can be built in $O(n + m)$ time. The total number of vertices in the block tree is $O(n)$. A linear time bucket sort routine is used to sort degrees of c-vertices and b-vertices. The data structure in Rosenthal & Goldner [18] can be used to keep track of current degrees of vertices in $blk(G)$. Vertices in $blk(G)$ with the same degree are kept in a linked list. An array is used to store the first element of each linked list. A vertex of the largest degree can be found in constant time and the position, in the array of linked list, of a vertex in the path found in step 6 can also be updated in constant time. To implement step 6, algorithms in Harel & Tarjan [10] and Schieber & Vishkin [19] are used to find the path P between two vertices v and w in $O(|P|)$ time. By Fact 1, the number of times a vertex is visited is no more than its degree. Since the summation of degrees of all vertices in a tree with n vertices is $O(n)$, the claim is true. $\square$

In the rest of this section we describe an efficient parallel algorithm for stage 3. Recall that the sequential algorithm adds one edge at a time and keeps adding edges until the block tree becomes a single vertex. In our parallel algorithm, however, we will find several pairs of leaves such that the path between any such pair of leaves passes through all critical c-vertices, if any. Thus the degrees of critical vertices in the new block tree decrease by the number of edges added to the original block tree. These pairs also satisfy the leaf-connecting condition (Definition 3), which guarantees that the number of leaves in the new block tree decreases by twice the number of edges added. The following Lemma 3 tells us that the addition of several edges in parallel as outlined above is a valid strategy.

Lemma 3 *Let G be a graph whose block graph is balanced and let G' be the graph obtained from G by adding a set of k edges $\mathcal{A} = \{(s_1,t_1), (s_2,t_2),\cdots,(s_k,t_k)\}$. For each i, $0 \leq i \leq k$, let G_i be the graph obtained from G by adding the set of edges $\{(s_1,t_1),\cdots,(s_i,t_i)\}$. Let s'_i and t'_i, $0 \leq i \leq k$, be the b-vertices in $blk(G_i)$ whose corresponding blocks contain s_i and t_i, respectively. If the path between s'_{i+1} and t'_{i+1}, $\forall i$, $1 \leq i < k$, in $blk(G_i)$ passes through all critical vertices in $blk(G_i)$ and s'_{i+1} and t'_{i+1} satisfy the leaf-connecting condition in G_i, then $blk(G')$ remains balanced and the value of the lower bound given in Theorem 1 applied to G' is k less than the same lower bound applied to G.*

<u>*Proof:*</u> We always obtain the same block graph for G_k no matter in what sequence we choose to add these k edges, since there is an unique block graph for each graph G. Thus $blk(G')$ $= blk(G_k)$. Since $blk(G_i)$, $\forall i$, $1 \leq i \leq k$, is balanced, $blk(G')$ is balanced. We know that the value of lower bound given in Theorem 1 applied to G_i is 1 less than the value of the same lower bound applied to G_{i-1}, $\forall i$, $1 \leq i \leq k$, where $G_0 = G$. Hence the value of the lower bound given in Theorem 1 applied to G' is k less than the value of the same lower bound applied to G. $\square$

From Theorem 1 and Claim 1, we know that exactly $\lceil \frac{l}{2} \rceil$ edges must be added to biconnect G if $blk(G)$ is balanced. That is, we have to eliminate l leaves during the computation. Our parallel algorithm runs in stages with at least $\frac{1}{4}$ of the current leaves eliminated in parallel time $O(\log n)$ using a linear number of processors during each stage. We call this subroutine $O(\log n)$ times to complete the augmentation.

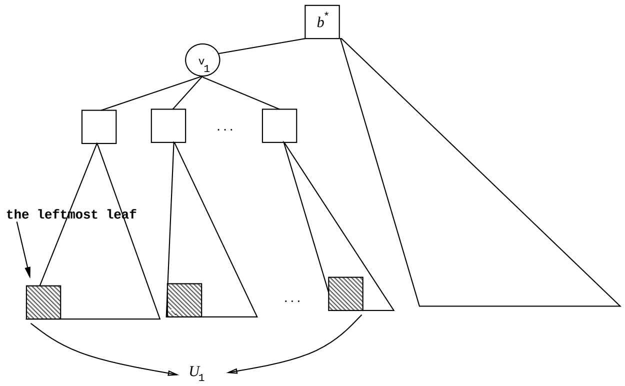

Figure 4: Each shadowed rectangle represents the leftmost leaf in a subtree rooted a child of v_1. Leaves in U_1 consist of leftmost leaves in every subtree rooted a child of v_1.

Recall that $a_i + 1$ is equal to the degree of the ith c-vertex v_i and $a_i \geq a_{i+1}$. T' is the subtree obtained from T by deleting the subtree rooted at v_1. Let $U_i = \{u|u$ is the leftmost leaf of T_y, where y is a child of $v_i\}$. For example, the leaves in U_1 are illustrated as shadowed rectangles in Figure 4.

Depending on the degree distribution of vertices in the block tree, the parallel algorithm for stage 3 is divided into 2 cases. In case one, $a_1 > \frac{l}{4}$. We have a c-vertex with a high degree. We pick the first $min\{a_1 - 1, \lceil\frac{l}{2}\rceil - a_3\}$ leaves in U_1 and call them W_1. Leaves in W_1 are matched with the first $min\{|W_1|, |U_2| - 1\}$ leaves in U_2. Unmatched leaves in W_1, if any, are matched with all remaining leaves but one in T' and finally properly matched within themselves, if necessary. In case two, $a_1 \leq \frac{l}{4}$. There is no c-vertex with a large degree. We show that we can find a vertex u^* with approximately the same number of leaves in each subtree rooted at a child of u^*. If u^* is a b-vertex, a suitable number of leaves between subtrees rooted at children of u^* are matched. Otherwise, u^* is a c-vertex and a suitable number of subtrees rooted at children of u^* are first merged into a single subtree rooted at u^*. Then leaves in the merged subtree are matched with leaves outside.

The algorithm first finds the matched pairs of leaves in each case. Then we add edges between matched pairs of leaves and update the block tree at the end of each case. The block tree and the sequence of cutpoints $v_1, \cdots, v_{n_c}$ will not be changed during the execution of each case.

We now describe the two cases in detail.

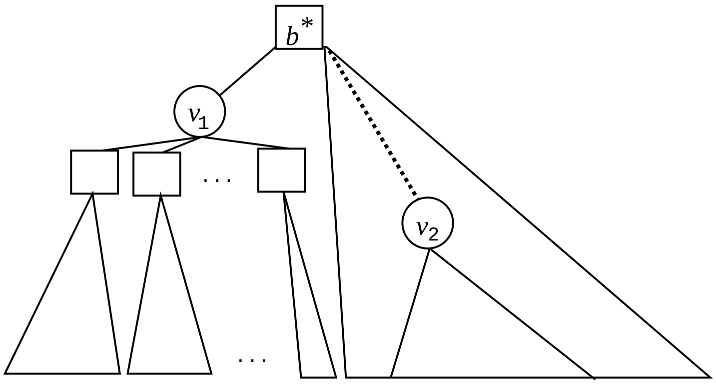

Figure 5: A normalized tree. Vertex v_1 is a c-vertex with the largest degree. Vertex v_2 is a c-vertex with a degree larger than or equal to any other c-vertices in $T - v_1$. We permute the children of v_1 in non-increasing order (from left to right) of the number of leaves in subtrees rooted at them.

4.3.1 Case 1: $a_1 > \frac{l}{4}$

We root the block tree at the b-vertex b^* which is adjacent to v_1 and is on the path from v_1 to v_2. Let v_1 be the leftmost child of b^*. We permute the children of v_1 in non-increasing order (from left to right) of the number of leaves in subtrees rooted at them. We will call this procedure *tree-normalization* and the resulting tree T. Figure 5 illustrates a normalized tree.

Recall that U_1 is the set of leftmost leaves in subtrees rooted at children of v_1. We select the first (from left to right) $min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\}$ leaves from U_1 and call the set W_1. The order of the leaves as specified in the original tree is preserved. There are four phases for this case. In phases 1 and 2, leaves in W_1 are matched with leaves not in T_1. In phase 3, leaves in W_1 are matched with leaves in T_1 excluding those in W_1. In phase 4, the remaining leaves in W_1 are matched between themselves. The algorithm executes each phase in turn once until there is no leaf in W_1 left to be matched.

We now describe the four phases in detail. After the description, we give the overall parallel algorithm for case 1 and prove that it eliminates a constant fraction of the leaves while maintaining the lower bound described in Theorem 1.

Phase 1: All leaves but the rightmost one in U_2 are matched with the rightmost $a_2 - 1$ leaves of W_1. The matched leaves are removed from W_1. An example of the pairs of leaves matched in phase 1 is given in Figure 6.

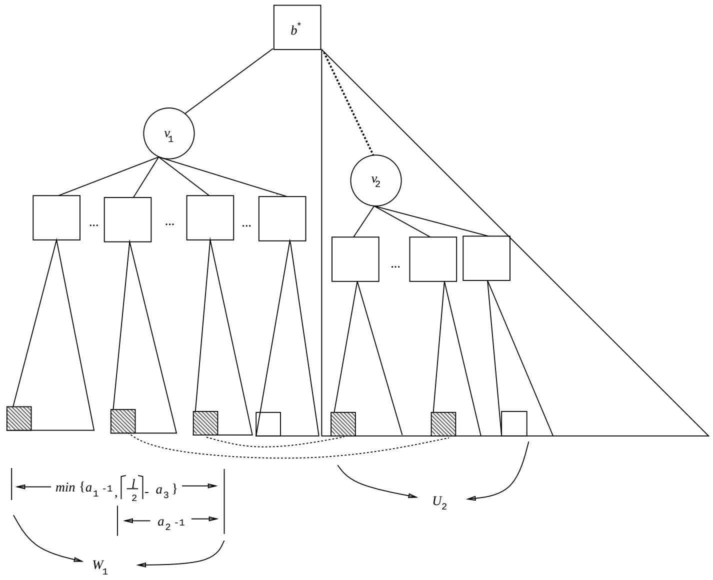

Figure 6: Pairs of matched leaves found in phase 1 of case 1 are connected by dotted lines. W_1 consists of the leftmost leaves from the first $min\{a_1-1,\lceil\frac{l}{2}\rceil-a_3\}$ subtrees rooted at children of v_1. All of the leftmost leaves in subtrees rooted at children of v_2 are in the set U_2, and all except the rightmost leaf in U_2 are matched (if possible).

set of pairs of vertices function phase1(**modifies set of vertices** W_1, U_2);
    set of pairs of vertices L; **vertex** u, v;
    $L := \{\}$;{$*$ L is the set of matched pairs. $*$}
    number leaves in W_1 from right to left starting from 1;
    number leaves in U_2 from left to right starting from 1;
    $k := min\{|U_2|-1, |W_1|\}$;
    pfor $i = 1\ ..\ k \rightarrow$
        $u :=$ the ith leaf in W_1; remove u from W_1;
        $v :=$ the ith leaf in U_2; remove v from U_2;
        $L := L \cup \{(u, v)\}$
    rofp;
    return L
end phase1;

Phase 2: We match all remaining leaves but one in T' with the rightmost leaves of W_1 and remove matched leaves from W_1. An example of the pairs of leaves matched in phase 2

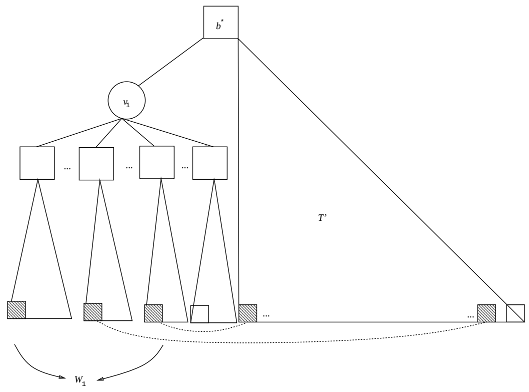

Figure 7: Pairs of matched leaves found in phase 2 of case 1 are connected by dotted lines. Recall that T_1 is the subtree of T rooted at v_1. T' is the subtree of T obtained by deleting T_1. W_1 consists of the leftmost leaves in the first $min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\} - a_2 + 1$ subtrees rooted at children of v_1. Leaves in W_1 are matched with all but the rightmost leaf in T'.

is given in Figure 7.

set of pairs of vertices function phase2(**modifies set of vertices** W_1,**tree** T');
  set of pairs of vertices L; **vertex** u,v;
  $L := \{\}$;{$*$ L is the set of matched pairs. $*$}
  number leaves in W_1 from right to left starting from 1;
  number leaves in T' from left to right starting from 1;
  $k := min\{$the number of leaves in T' minus 1, $|W_1|\}$;
  pfor $i = 1 .. k \rightarrow$
    $u :=$ the ith leaf in W_1; remove u from W_1; $v :=$ the ith leaf in T';
    $L := L \cup \{(u, v)\}$
  rofp;
  return L
end phase2;

Phase 3: Recall that T is the original block tree before phase 1, l is the number of leaves in T, v_1 is a c-vertex with the largest degree in T, T_1 is the subtree of T rooted at v_1, l_1 is the number of leaves in T_1, T' is the tree obtained from T by removing T_1 and $U_1 = \{u | u$ is the leftmost leaf of T_y, where y is a child of $v_1\}$. Note that there are $min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\}$ $-(l - l_1 - 1)$ leaves remaining in W_1. Leaves in W_1 come from the first $|W_1|$ members (from

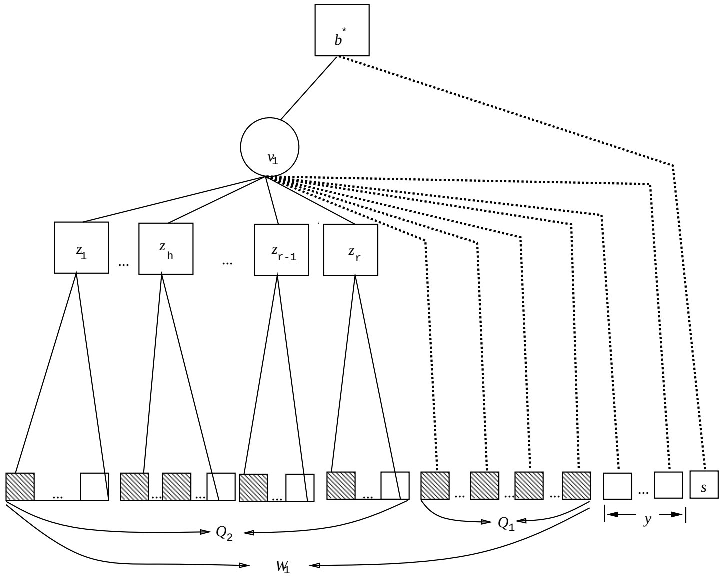

Figure 8: Notations used in the proof of a claim used in phase 3 of case 1. The tree shown is T^*, the updated block tree obtained by adding edges between pairs of matched leaves found in phase 1 and phase 2. Q_2 consists of all but the rightmost leaf in each subtree rooted at a child of v_1. Q_1 consists of v_1-chain leaves in W_1 after phase 2. The number of subtrees rooted at a child of v_1 with more than one leaf is r. the number of v_1-chain leaves not in Q_1 is y.

left to right) of U_1. Let the set of v_1-chain leaves in W_1 be Q_1. We denote by Q_2 the set of leaves other than the rightmost one of each subtree rooted at a child of v_1. (Note that $Q_1 \cap Q_2 = \emptyset$.) In this phase, we match all leaves in Q_1 (i.e., all v_1-chain leaves in W_1) with an equal number of leaves in Q_2. Leaves in W_1 that are matched in phase 3 (Q_1 and $W_1 \cap Q_2$) are removed from W_1.

Claim 3 shows that we can always find enough leaves in Q_2 to match all leaves in Q_1.

Claim 3 $|Q_2| \geq |Q_1|$, *if* $l > 3$.

Proof: If $|Q_1| = 0$, the claim is true. Let $|Q_1| > 0$. Recall that there is only one unmatched leaf s left in T' after phase 2. Let T^* be the block tree obtained from T by adding edges between matched pairs of leaves found in phase 1 and phase 2. We root T^* at the b-vertex b^* which is adjacent to v_1 and is on the path from v_1 to s. Let r be the number of subtrees rooted at a child of v_1 in T^* with more than one leaf. Let y be the number of v_1-chain leaves not in Q_1. The notations used in this proof are shown in Figure 8.

The total number of leaves in T^* is equal to $|Q_1| + |Q_2| + r + y + 1$ if $|Q_1| > 0$. The degree of v_1 in T^* is equal to $|Q_1| + r + y + 1$. Since T^* is balanced (for a proof, see Claim 5 at the end of this section), v_1 is not massive and hence

$$|Q_1| + r + y \leq \lceil \frac{|Q_1| + |Q_2| + r + y + 1}{2} \rceil.$$

Thus

$$\begin{aligned} & 2|Q_1| + 2r + 2y \leq |Q_1| + |Q_2| + r + y + 2 \\ \Rightarrow \quad & |Q_1| + r + y - 2 \leq |Q_2| \end{aligned}$$

We know that $r \geq 1$, otherwise v_1 is massive if $l > 3$. It is also true that $y \geq 1$ if $|Q_1| > 0$. Thus $|Q_1| \leq |Q_2|$. □

The procedure for phase 3 is described below.

set of pairs of vertices function phase3(**modifies set of vertices** Q_1, Q_2);
  set of pairs of vertices L; **vertex** u, v;
  $L := \{\}$;{$*$ L is the set of matched pairs. $*$}
  number leaves in Q_2 from right to left starting from 1;
  number leaves in Q_1 from 1 to $|Q_1|$ in arbitrary order;
  $k := |Q_1|$;
  pfor $i = 1 \,..\, k \rightarrow$
    $u :=$ the ith leaf in Q_2; remove u from Q_2;
    $v :=$ the ith leaf in Q_1; remove v from Q_1;
    $L := L \cup \{(u, v)\}$
  rofp;
  return L
end phase3;

Phase 4: The remaining leaves of W_1 that are not matched during phase 3 are matched within themselves. If the number of remaining leaves in W_1 is odd, we match one of them with the rightmost leaf in the subtree rooted at v_1. An example of the pairs of leaves matched in phase 4 is given in Figure 9.

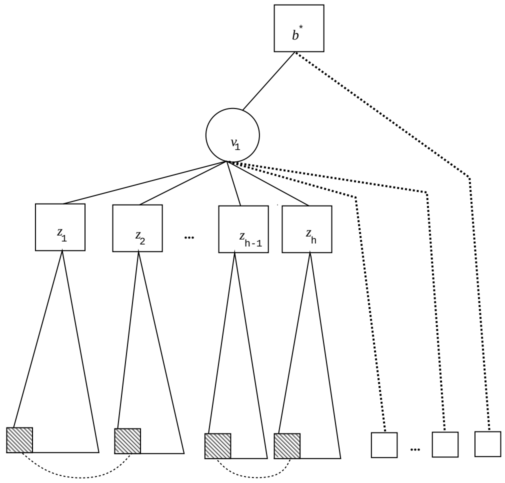

Figure 9: Illustrating phase 4 of case 1. The remaining leaves in W_1 are matched within themselves.

set of pairs of vertices function phase4(**modifies set of vertices** W_1,**tree** T);
  set of pairs of vertices L; **vertex** u, v;
  $L := \{\};\{*$ L is the set of matched pairs. $*\}$
  number leaves in W_1 in arbitrary order from 1 to $|W_1|$;
  $k := \lceil \frac{|W_1|}{2} \rceil$;
  pfor $i = 1 \,..\, k \rightarrow$
    $u :=$ the $(2 * i - 1)$th leaf in W_1; remove u from W_1;
    if $2 * i \leq |W_1| \rightarrow v :=$ the $(2 * i)$th leaf in W_1; remove v from W_1
    $|$ $2 * i > |W_1| \rightarrow v :=$ the rightmost leaf in the subtree rooted at v_1
    fi;
    $L := L \cup \{(u, v)\}$
  rofp;
  return L
end phase4;

We now describe our algorithm for case 1.

set of pairs of vertices function case1(**tree** T);
  set of pairs of vertices L; **set of vertices** W_1, Q_1, Q_2;**vertex** b^*; **tree** T';
  root T at the b-vertex b^* which is adjacent to v_1 and is on the path from v_1 to v_2;
  let v_1 be the leftmost child of b^* in T;

permute the children of v_1 in non-increasing order (from left to right)
of the number of leaves in subtrees rooted at them;
W_1 := the first (from left to right) $min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\}$ leaves of U_1;
L := phase1(W_1, U_2); {* L is the set of matched pairs. *}
if $W_1 \neq \{\} \rightarrow L := L \cup$ phase2(W_1, T') **fi;**
T'_1 := the subtree of T_1 with the first $|W_1|$ subtrees rooted at children of v_1;
Q_1 := the set of v_1-chain leaves in T_{v_1};
$Q_2 := \{u | u$ is a non-leftmost leaf of T_y, where T_y has more than 1 leaf
and y is a child of v_1 in $T'_1\}$;
if $W_1 \neq \{\} \rightarrow L := L \cup$ phase3(Q_1, Q_2) **fi;** $W_1 := W_1 \cap Q_2$;
if $W_1 \neq \{\} \rightarrow L := L \cup$ phase4(W_1,T) **fi;**
return L
end case1;

Claim 4 *The number of matched pairs k in case 1 satisfies $\lceil \frac{l}{2} \rceil - a_3 \geq k \geq \frac{l}{8}$, if $l > 3$.*

Proof: Let $z = min\{a_1 - 1, \lceil \frac{l}{2} \rceil - a_3\}$. If the procedure does not execute phase 3 and 4, we match z pairs. Because $a_1 - 1 \geq \frac{l}{4}$ and $\lfloor \lceil \frac{l}{2} \rceil - a_3 \rfloor \geq \lfloor \frac{l}{8} \rfloor$ for $l > 3$ (Corollary 1), we know that $z \geq \frac{l}{8}$, if $l > 3$. Otherwise in the worst case, we match only $a_2 - 1$ pairs during phase 1 and phase 2. A pair of vertices matched during phase 3 or 4 might be both members of W_1. Thus

$$k \geq a_2 - 1 + \lceil \frac{z - a_2 + 1}{2} \rceil \geq \frac{z + a_2 + 1}{2}.$$

If $z = \lceil \frac{l}{2} \rceil - a_3$, then $k \geq \frac{\lceil \frac{l}{2} \rceil - 1}{2}$, which is greater than or equal to $\frac{l}{8}$ if $l > 3$ and k is an integer. If $z = a_1 - 1$, then $k \geq \lceil \frac{a_1 - 1}{2} \rceil$. Because $a_1 > \frac{l}{4}$, $k \geq \frac{l}{8}$. □

Claim 5
(1) Each pair of matched vertices found in function case1 satisfies the leaf-connecting condition (Definition 3).
(2) Let us place an edge between each matched pair found in function case1 sequentially and update the block graph each time we add an edge. Critical vertices, if any, of the block graph are on the path between the endpoints of each edge placed.

Proof: From part (4) in Fact 1, degrees of v_1 and v_2 decrease only by 1 by adding an edge between a pair of vertices matched. Let us consider paths between pairs of vertices matched in each phase. We show that we can find at least two vertices with degrees more than 2 in each path.

Phase 1: The path between each pair of matched vertices passes through v_1 and v_2 whose degrees are at least 3.

Phase 2: The path between each pair of matched vertices passes through v_1 and the root b^* whose degrees are at least 3.

Phase 3 and phase 4: The path P between each pair of matched vertices passes through v_1 whose degree is at least 3. P also passes through a child u of v_1 where the subtree rooted at u has more than one leaf. Thus the degree of u is more than 2.

Thus the leaf-connecting condition (Definition 3) holds for each pair of matched vertices.

Because we only add $min\{a_1-1,\lceil\frac{l}{2}\rceil-a_3\}$ edges during case 1, v_3 and thus v_i, $\forall i$, such that $i \geq 4$, does not become critical. From the previous discussion, the path between each pair of matched vertices passes through v_1 and v_2, the only two possible critical vertices, during phase 1. We reduce degrees of possible critical vertices by one by adding one new edge between each pair of matched vertices. If we match any pair of vertices after phase 1, the degree of v_2 is at most 2 and the degree of v_1 is at least 3. Thus v_1 is the only possible critical vertex. The path between each pair of matched vertices passes through v_1 after phase 1. We reduce the degree of the possible critical vertex, v_1, by one by adding one new edge between any pair of matched vertices. Thus the claim is true. □

Corollary 6 *Let k be the number of matched pairs found in function case1. Let G' be the resulting graph obtained from the current graph G by adding a new edge between each matched pair of leaves. The value of the lower bound given in Theorem 1 applied to G' is k less than the value of the same lower bound applied to G and $blk(G')$ remains balanced. Let l be the number of leaves in $blk(G)$. The number of leaves in $blk(G')$ is at most $\frac{3l}{4}$, if $l > 3$.*

Proof: From part (1) in Claim 5, the number of leaves in $blk(G')$ is $2k$ less than the number of leaves in $blk(G)$. Since $k \geq \frac{l}{8}$ if $l > 3$ (Claim 4), the number of leaves in $blk(G')$ is at most $\frac{3l}{4}$ if $l > 3$. From part (2) in Claim 5, the block graph of each intermediate graph remains balanced even if we place a new edge between each matched pair of leaves found in function case1 sequentially. By Lemma 3, we know that the value of the lower bound given in Theorem 1 applied to G' is k less than the value of the same lower bound applied to G and $blk(G')$ remains balanced. □

4.3.2 Case 2: $a_1 \leq \frac{l}{4}$

In this case, we take advantage of the fact that no c-vertex has a large degree. Because there is no critical c-vertex, the algorithm can add at least $\lceil \frac{l}{2} \rceil - a_1$ edges between leaves that satisfy the leaf-connecting condition (Definition 3) without worrying about whether the path between them passes through a critical c-vertex. This gives a certain degree of freedom for us to choose the matched pairs. We first root the block tree such that no subtree other than the one rooted at the root has more than half of the total number of leaves.

Given any rooted tree T, we use l_v to denote the number of leaves in the subtree rooted at a vertex v. The following lemma shows that we can reroot T at a vertex u^* such that no subtree rooted at a child of u^* has more than half of the total number of leaves.

Lemma 4 *Given a rooted tree T, there exists a vertex u^* in T such that $l_{u^*} > \frac{l}{2}$, but none of the subtrees rooted at children of u^* has more than $\frac{l}{2}$ leaves.*

Proof: We permute children of each non-leaf vertex v from left to right in non-increasing order of the number of leaves in the subtrees rooted at them. Let us consider the leftmost path P of the tree T. It is obvious that there exists such a vertex u^* in P. $\square$

We root the block tree at u^* and permute children of u^* from left to right in non-increasing order of the number of leaves in subtrees rooted at them. Let the rooted tree be T. Let u_i, $\forall i$, $1 \leq i \leq r$, be the children (from left to right) of u^* and x_i be the number of leaves in the subtree rooted at u_i. Note that $x_i \leq \frac{l}{2}$, for each i. There are two subcases depending on whether u^* is a b-vertex or a c-vertex. We describe the two subcases in detail in the following paragraphs.

Subcase 2.1: u^* is a b-vertex

We show that we can partition subtrees rooted at children of the root into two sets "evenly" such that we can match leaves between the two partitions. We first give a claim to show how to perform the partition.

Claim 6 *There exists p, such that $1 \leq p < r$ and $\frac{l}{2} \geq \sum_{i=1}^{p} x_i > \frac{l}{4}$.*

Proof: We know that $x_i \geq x_{i+1}$, $\forall i$, $1 \leq i < r$, and $x_i \leq \frac{l}{2}$, $\forall i$, $1 \leq i \leq r$. Thus $\exists$p$, 1 \leq p < r$, such that

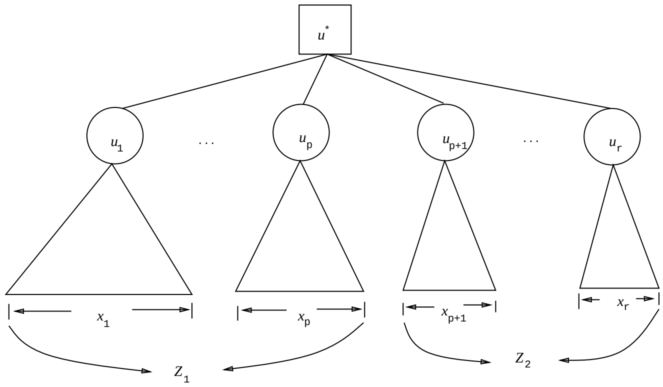

Figure 10: The notations used in case 2.1. The number of leaves in the subtree rooted at u_i is x_i. We find the largest p such that the total number of leaves in the first p subtrees rooted at children of the root is greater than $\frac{l}{4}$, but less than or equal to $\frac{l}{2}$. Leaves in the first p subtrees rooted at children of b^* are in Z_1. Z_2 consists of the rest of the leaves in the tree.

$$\sum_{i=1}^{p} x_i \leq \frac{l}{2} \text{ and } \sum_{i=1}^{p+1} x_i > \frac{l}{2}.$$

Because $x_i \geq x_{i+1}$, $\forall i$, $1 \leq i < r$, we know that $\sum_{i=1}^{p} x_i > \frac{1}{2}(\frac{l}{2})$. $\square$

The notations used for this subcase are illustrated in Figure 10.

Corollary 7 $\sum_{i=1}^{p} x_i \leq l - \sum_{i=1}^{p} x_i$.

We match $min\{(\sum_{i=1}^{p} x_i) - 1, \lceil \frac{l}{2} \rceil - a_1\}$ leaves in subtrees T_{u_i}, $\forall i$, $1 \leq i \leq p$, with leaves outside them. From Claim 6 and Corollary 7, we know that the matching can be done.

Corollary 8 *The number of matched pairs k in case 2.1 satisfies $\lceil \frac{l}{2} \rceil - a_1 \geq k > \frac{l}{4}$, if $l > 3$.*

set of pairs of vertices function case2_1(**tree** T);
{$*$ l is the number of leaves in T. $*$}
  integer p; **set of pairs of vertices** L; **set of vertices** Z_1, Z_2; **vertex** u,v;
  let u_i be the ith (from left to right) child of the root;
  let x_i be the number of leaves in the subtree rooted at u_i;
  find the largest integer p such that $\sum_{i=1}^{p} x_i \leq \frac{l}{2}$, but $\sum_{i=1}^{p+1} x_i > \frac{l}{2}$;

$L := \{\};\{*\ L$ is the set of matched pairs. $*\}$
$Z_1 :=$ the set of leaves in the subtrees rooted at u_i, $\forall i$, $1 \leq i \leq p$;
$Z_2 :=$ the set of leaves in the subtrees rooted at $u_i, i > p$;
number leaves in Z_1 in arbitrary order from 1 to $|Z_1|$;
number leaves in Z_2 in arbitrary order from 1 to $|Z_2|$;
$k := min\{(\sum_{i=1}^{p} x_i) - 1, \lceil \frac{l}{2} \rceil - a_1\}$;
pfor $i = 1 \,..\, k \rightarrow$
  $u, v :=$ the ith vertex in Z_1 and Z_2, respectively;
  $L := L \cup \{(u, v)\}$
rofp;
return L
end case2_1;

Claim 7 *Any matched pair found in function case2_1 satisfies the leaf-connecting condition (Definition 3), if* $l > 3$.

Proof: Consider the path P between a pair of matched leaves u and v. Let u be a leaf in a subtree rooted at a u_x, $1 \leq x \leq p$, and let v be a leaf in a subtree rooted at a u_y, $p < y \leq r$. Since we match $min\{|Z_1| - 1, \lceil \frac{l}{2} \rceil - a_1\}$ leaves in Z_1 with an equal number of leaves in Z_2 and $|Z_1| \leq |Z_2|$ (Corollary 7), there is at least one leaf in a subtree rooted at a u_i, $1 \leq i \leq p$, that is not matched and there is also another leaf in a subtree rooted at a u_j, $p < j \leq r$, that is not matched if $l > 3$. The path P contains the root. If the degree of the root is at least 4, u and v satisfy the leaf-connecting condition (Definition 3). If the degree of the root is 3, P contains either u_i or u_j, whose degree is at least 3. Otherwise, P contains both u_i and u_j whose degrees are at least 3. Thus u and v satisfy the leaf-connecting condition (Definition 3). □

Corollary 9 *Let k be the number of matched pairs found in function case2_1. Let G' be the resulting graph obtained from the current graph G by adding a new edge between each matched pair of leaves. The value of the lower bound given in Theorem 1 applied to G' is k less than the value of the same lower bound applied to G and $blk(G')$ remains balanced. Let l be the number of leaves in $blk(G)$. The number of leaves in $blk(G')$ is at most $\frac{l}{2}$, if $l > 3$.*

Proof: From Claim 7, the number of leaves in $blk(G')$ is $2k$ less than the number of leaves in $blk(G)$. Since $k \geq \frac{l}{4}$ if $l > 3$ (Corollary 8), the number of leaves in $blk(G')$ is at most $\frac{l}{2}$. From Corollary 8, we add at most $\lceil \frac{l}{2} \rceil - a_1$ edges, thus no c-vertex in $blk(G')$ becomes massive. By Lemma 3, we know that the value of the lower bound given in Theorem 1 applied to G' is k less than the value of the same lower bound applied to G and $blk(G')$ remains balanced. $\square$

Subcase 2.2: u^* is a c-vertex

Recall that the u_i, $\forall i$, $1 \leq i \leq r$, are the children (from left to right) of u^* (the root). Let x_i be the number of leaves in the subtree rooted at u_i. We know that $\frac{l}{2} \geq x_i$, $\forall i$, $1 \leq i \leq r$, and $x_i \geq x_{i+1}$, $\forall i$, $1 \leq i < r$.

We partition the set of subtrees rooted at children of the root into two sets such that we can match leaves between two sets. We first give a claim to show how to partition the set of subtrees.

Claim 8 *Let q be the largest integer with $x_q \geq 2$. There exists an integer p such that $1 \leq p \leq q$ and $\frac{l}{2} \geq \sum_{i=1}^{p} x_i > \frac{l}{8} + (p-1)$.*

Proof: If $x_1 > \frac{l}{8}$, then $p = 1$. If $x_1 \leq \frac{l}{8}$, we can find an integer p such that $\frac{l}{2} \geq \sum_{i=1}^{p} x_i > \frac{3l}{8}$ using an argument similar to the one given in the proof of Claim 6. By definition, we know that $x_p \geq 2$ because otherwise the root (a c-vertex) is massive. Thus $p \leq \frac{l}{4}$. Hence $(\sum_{i=1}^{p} x_i) - (p-1) > \frac{l}{8}$. $\square$

Let T_{u_i} be the subtree rooted at u_i. We define the *merge* operation for the collection of subtrees T_{u_i}, $\forall i$, $1 \leq i \leq p$, as follows. We first connect the rightmost leaf of T_{u_i} and the leftmost leaf of $T_{u_{i+1}}$, $\forall i$, $1 \leq i < p$. This can be done by the fact that each T_{u_i}, $\forall i$, $1 \leq i \leq p$, has at least 2 leaves.

Claim 9 *Let T^* be the block tree obtained from T by collapsing b-vertices that are in the same fundamental cycle created by the addition of new edges introduced by the merge operation.*
(1) The merge operation creates only one b-vertex b^*.
(2) Vertex b^* is a child of the root and b^* is the root of the subtree that contains the updated portion of the block tree.

Proof: Let C_i, $\forall i$, $1 \leq i < p$, be the fundamental cycle created by connecting the rightmost leaf of T_{u_i} and the leftmost leaf of $T_{u_{i+1}}$. The cycles C_i and C_{i+1}, $\forall i$, $1 \leq i < p-1$, share the

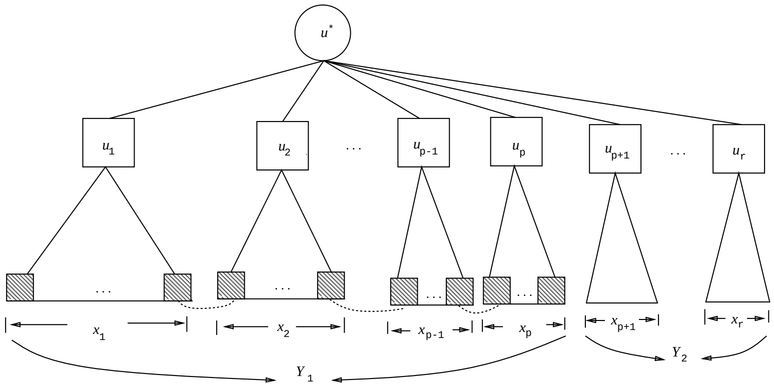

Figure 11: The notations used in case 2.2. The number of leaves in the subtree rooted at u_i is x_i. We find the largest p such that the total number of leaves in the first p subtrees rooted at children of the root is greater than $\frac{l}{8} + (p-1)$, but at most $\frac{l}{2}$. We first merge subtrees rooted at u_i, $\forall i$, $1 \le i \le p$, by connecting the rightmost leaf in the subtree rooted at u_i and the leftmost leaf in the subtree rooted at u_{i+1}, $\forall i$, $1 \le i < p$. Leaves in the first p subtrees rooted at children of u^* are in Y_1. Y_2 consists of the rest of leaves in the tree. We then match $min\{(\sum_{i=1}^{p} x_i) - 1, \lceil \frac{l}{2} \rceil - a_1\} - (p-1)$ leaves in Y_1 with leaves in Y_2.

b-vertex u_i. From part (2) in Fact 1, we know that all b-vertices in cycles C_i, $\forall i$, $1 \le i < p$, shrink into a single b-vertex in the new block tree. Let this new b-vertex be b^*. Thus part (1) of the claim is true. Part (2) of the claim follows from part (4) in Fact 1. □

Note that if we root the updated block tree T^* given in Claim 9 at the b-vertex b^*, the situation is similar to that in case 2.1. Thus we can match an additional $min\{(\sum_{i=1}^{p} x_i) - 1, \lceil \frac{l}{2} \rceil - a_1\} - (p-1)$ pairs of vertices by pairing up unmatched leaves in subtrees T_{u_i}, $\forall i$, $1 \le i \le p$, and leaves in subtrees in subtrees T_{u_i}, $\forall i$, $p < i \le r$. This procedure is given below in case2_2. The notations used are shown in Figure 11.

Corollary 10 *The number of matched pairs k in case 2.2 satisfies $\lceil \frac{l}{2} \rceil - a_1 \ge k \ge \frac{l}{8}$, if $l > 3$.*

set of pairs of vertices function case2_2(**tree** T);
  vertex u,v; **integer** p; **set of vertices** Y_1, Y_2; **set of pairs of vertices** L;
  let u_i, $\forall i$, $1 \le i \le r$, be the children of the root u^*;
  let T_{u_i} be the subtree rooted at u_i; let x_i be the number of leaves in T_{u_i};
  find the largest integer p such that $\frac{l}{2} \ge \sum_{i=1}^{p} x_i > \frac{l}{8} + (p-1)$;
  Y_1 := the set of leaves in the subtrees rooted at u_i, $\forall i$, $1 \le i \le p$;

$Y_2 :=$ the set of leaves in the subtrees rooted at $u_i, i > p$;
$L := \{\}$; $\{*\ L$ is the set of matched pairs. $*\}$
pfor $i = 1\ ..\ p-1 \rightarrow$
  let u be the leftmost leaf of T_{u_i}; let v be the rightmost leaf of $T_{u_{i+1}}$;
  $L := L \cup\ \{(u,v)\}$; remove u and v from Y_1
rofp;
number the leaves in Y_1 in arbitrary order from 1 to $|Y_1|$;
number the leaves in Y_2 in arbitrary order from 1 to $|Y_2|$;
$k := min\{\sum_{i=1}^{p} x_i, \lceil \frac{l}{2} \rceil - a_1\} - (p-1)$;
pfor $i = 1\ ..\ k \rightarrow$
  $u, v :=$ the ith vertex in Y_1 and Y_2, respectively;
  $L := L \cup \{(u,v)\}$
rofp;
return L
end case2_2;

Claim 10 *Each pair of vertices matched in function case2_2 satisfies the leaf-connecting condition (Definition 3), if $l > 3$.*

Proof: By Claim 9 and similar arguments given in the proof of Claim 7. □

Corollary 11 *Let k be the number of matched pairs found in function case2_2. Let G' be the resulting graph obtained from the current graph G by adding a new edge between each matched pair of leaves. The value of the lower bound given in Theorem 1 applied to G' is k less than the value of the same lower bound applied to G and $blk(G')$ remains balanced. Let l be the number of leaves in $blk(G)$. The number of leaves in $blk(G')$ is at most $\frac{3l}{4}$, if $l > 3$.*

Proof: From Claim 10, the number of leaves in $blk(G')$ is $2k$ less than the number of leaves in $blk(G)$. Since $k \geq \frac{l}{8}$ if $l > 3$ (Corollary 10), the number of leaves in $blk(G')$ is at most $\frac{3l}{4}$. From Corollary 10, we add at most $\lceil \frac{l}{2} \rceil - a_1$ edges, thus no c-vertex in $blk(G')$ becomes massive. By Lemma 3, we know that the value of the lower bound given in Theorem 1 applied to G' is k less than the value of the same lower bound applied to G and $blk(G')$ remains balanced. □

The complete procedure for case 2 is shown below.

set of pairs of vertices function case2(**tree** T);
$\{*$ l is the number of leaves in T; $a_1 + 1$ is the largest degree of all c-vertices in T. $*\}$
  vertex u^*;
  root T at an arbitrary vertex;
  find a vertex u^* such that there are more than $\frac{l}{2}$ leaves in the subtree rooted at u^*,
  but none of the subtrees rooted at a child of u^* have more than $\frac{l}{2}$ leaves;
  root T at u^*;
  permute children of u^* (from left to right) in non-increasing order of
  the number of leaves in subtrees rooted at them;
  if u^* is an b-vertex $\rightarrow$ **return** case2_1(T) | u^* is a c-vertex $\rightarrow$ **return** case2_2(T) **fi**
end case2;

The correctness of this algorithm is shown earlier in the two subcases (Corollary 9 and Corollary 11).

5 The Complete Parallel Algorithm and Its Implementation

We first describe the overall parallel algorithm and then an efficient parallel implementation on an EREW PRAM.

5.1 The Complete Parallel Algorithm

We are ready to present the complete parallel algorithm for the biconnectivity augmentation problem.

graph function par_bca(**graph** G);
$\{*$ The input graph G has at least 3 vertices; l is the number of leaves in the block graph T.
$*\}$
  set of pairs of vertices L; **tree** T; **vertex** u, v, α, β; **set of edges** S;
  $T := blk(G)$;
  if T is a forest $\rightarrow$ perform the procedure specified in Section 4.1 **fi**;
  if T is not balanced $\rightarrow$ perform the procedure specified in Section 4.2 **fi**;
  do $l \geq 2 \rightarrow$

if $l > 3 \rightarrow$ **if** $a_1 > \frac{l}{4} \rightarrow L :=$ case1(T) | $a_1 \leq \frac{l}{4} \rightarrow L :=$ case2(T) **fi**
| $l \leq 3 \rightarrow$ let u and v be two leaves in T; $L := \{(u,v)\}$
fi;
$S := \{\}$;
pfor each $(u,v) \in L \rightarrow$
find a non-cutpoint vertex α in the corresponding block of G represented by u;
find a non-cutpoint vertex β in the corresponding block of G represented by v;
add an edge between α and β; $S := S \cup \{(u,v)\}$
rofp;
1. $T :=$ par_update(T,S) {* The procedure par_update returns the updated block tree after adding the set of edges in S. *}
od;
return G
end par_bca;

The correctness of algorithm par_bca follows from the correctness we established earlier of the various cases (Corollary 6, Corollary 9 and Corollary 11).

In the previous sections, we have shown details of each step in algorithm par_bca except step 1. We now describe an algorithm for updating the block tree given the original block tree T and the set of edges S added to it (step 1 in algorithm par_bca).

To describe the parallel algorithm for updating the block graph T after adding a set of edges S, we define the following equivalence relation $\mathcal{R}$ on the set of b-vertices B, where B=$\{v|v$ is a b-vertex in T and v is in a cycle created by adding the edges in $S\}$. A pair (x,y) is in $\mathcal{R}$ if and only if $x \in B$, $y \in B$ and vertices in blocks represented by x and y are in the same block after adding the edges in S. It is obvious that $\mathcal{R}$ is reflexive, symmetric and transitive. Since $\mathcal{R}$ is an equivalence relation, we can partition B into k disjoint subsets B_i, $1 \leq i \leq k$, such that for each i, $x, y \in B_i$ implies $(x,y) \in \mathcal{R}$ and for any $(x,y) \in \mathcal{R}$, x and y both belong to the same B_i.

The following claim can easily be verified by using Fact 1 and the above definition for the equivalence relation on the set of b-vertices.

Claim 11 *Two b-vertices b_1 and b_2 are in the same equivalence class if and only if there exists a set of fundamental cycles $\{C_0, \cdots, C_q\}$ such that $b_1 \in C_0$, $b_2 \in C_q$ and C_i and C_{i+1}*

share a common b-vertex, for $0 \le i < q$. $\square$

Notice that fundamental cycles in the block tree created by adding edges between pairs of leaves found in phase 1 and phase 2 of case 1 and subcase 2.1 share a common b-vertex (the root). Any pair of fundamental cycles created by adding edges between pairs of leaves found in phase 3 of case 1 either share a child of v_1 (a b-vertex) or do not share any b-vertex at all. Fundamental cycles created by adding edges between pairs of leaves found in phase 4 of case 1 do not share any b-vertex with any other fundamental cycle. Any pair of fundamental cycles created by adding edges between pairs of leaves found in subcase 2.2 share either the root (a b-vertex) or a b-vertex created by the merge operation (Claim 9).

From the above discussion, we know that b-vertices in fundamental cycles formed by adding edges due to phase 1 and phase 2 of case 1 shrink into a single b-vertex in the new block tree. The b-vertices in fundamental cycles formed by adding edges due to phase 3 of case 1 which share a common child of v_1 shrink into a single b-vertex. The b-vertices in a fundamental cycle formed by adding edges due to phase 4 of case 1 shrink into a single b-vertex. The b-vertices in all fundamental cycles formed by adding edges due to subcase 2.1 or subcase 2.2 shrink into a single b-vertex. Thus we know how to compute the equivalence classes of $\mathcal{R}$.

We now describe the algorithm for updating the block tree given the original block tree T and the set of edges S added to it.

tree function par_update(**tree** T,**set of edges** S);
  vertex w; **integer** k; **set of edges** S_1,S_2,S_3,S_4;
  let B be the set of b-vertices in a cycle in $T \cup S$;
  $\{*$ The partition $\{B_i | 1 \le i \le k\}$ of B is computed such that two b-vertices b_1 and b_2 are in the same set if and only if there exists a set of fundamental cycles $\{C_0, \cdots, C_q\}$ in $T \cup S$ with $b_1 \in C_0$, $b_2 \in C_q$ and C_i and C_{i+1} share a common b-vertex, $\forall 0 \le i < q$. $*\}$
  if S is constructed from pairs found in case 1 $\rightarrow$
    let S_i, $\forall i$, $1 \le i \le 4$, be the edges in S corresponding to the pairs found in phase i;
    let B_1 be the set of b-vertices in fundamental cycles in $T \cup S_1 \cup S_2$;
    pfor the ith child z_i of v_1 $\rightarrow$
      let B_{i+1} be the set of b-vertices in fundamental cycles in $T \cup S_3$ that contain z_i
    rofp;

k := 1 + the number of children of v_1 in T;
pfor the ith edge e_i in $S_4 \rightarrow$
let B_{i+k} be the set of b-vertices in the fundamental cycle in $T \cup \{e_i\}$
rofp;
$k := k + |S_4|$
| S is constructed from the pairs found in case 2 $\rightarrow$
$B_1 := B$; $k := 1$
fi;
pfor $i = 1 \,..\, k \rightarrow$
collapse all b-vertices in B_i into a single b-vertex
rofp;
eliminate parallel edges created by collapsing b-vertices;
let T' be this graph;
pfor each c-vertex w in $T' \rightarrow$ **if** degree(w) = 1 $\rightarrow$ eliminate w **fi** **rofp**;
return T'
end par_update;

Claim 12 *Function par_update returns the updated block tree.*

Proof: From Claim 11 and parts (3) and (4) in Fact 1. □

Note that we can get the updated block tree by using an algorithm for finding biconnected components. We will, however, show in Section 5.2 that the time needed on an EREW PRAM for updating the block tree using function par_update is less than what is needed to compute connected components using a linear number of processors. Hence we do not want to use the straightforward algorithm for finding connected components to implement function par_update.

5.2 The Parallel Implementation

We now describe an efficient parallel implementation for algorithm par_bca.

Given an undirected graph, we can find its block graph in time $O(\log^2 n)$ using a linear number of processors on an EREW PRAM by the parallel algorithm in Tarjan & Vishkin

[23] for finding biconnected components and using some procedures in Nath & Maheshwari [16].

The parallel versions of stage 1 and stage 2 are described in Section 4.1 and Section 4.2, respectively. In stage 3, the children-permutation procedure can be done in time $O(\log n)$ using a linear number of processors on an EREW PRAM by calling the parallel merge sort routine in Cole [2] and using the Euler tour technique in Tarjan & Vishkin [23] to restructure and normalize the tree. To perform functions case1, case2 and par_update, we need the following procedures.

- A procedure that numbers leaves in the tree from left to right or from right to left.
- For each vertex v in a tree, find the number and the set of leaves in the subtree rooted at v.
- For a vertex v in a tree, find the leftmost leaf of each subtree rooted at a child of v.
- For a tree T with a set of edges S added between leaves in T, compute:
  - the number of cycles that pass through a vertex in $T \cup S$;
  - the set of vertices in a cycle in $T \cup S$.

All of these procedures can be done in $O(\log n)$ time using a linear number of processors on an EREW PRAM by using the Euler technique in Tarjan & Vishkin [23] and procedures in Schieber & Vishkin [19].

From Corollary 6, Corollary 9 and Corollary 11, we know that algorithm par_bca removes at least a quarter of the leaves in the current block graph during each execution of the **do** loop. Initially, the number of leaves is at most n. Hence the main **do** loop in algorithm par_bca is executed $O(\log n)$ times. Each iteration takes $O(\log n)$ time using a linear number of processors, since the parallel sorting routine used in permuting children needs $O(n)$ processors. This establishes the following claim.

Claim 13 *The biconnectivity augmentation problem on an undirected graph can be solved in time $O(\log^2 n)$ using a linear number of processors on an EREW PRAM, where n is the number vertices in the input graph.*

6 Conclusion

In this paper we have presented a linear time sequential algorithm and an efficient parallel algorithm to find a smallest augmentation to biconnect a graph. Our sequential algorithm corrects an error in an earlier algorithm proposed for this problem in Rosenthal & Goldner [18]. Our parallel algorithm is new, and it runs in $O(\log^2 n)$ time using a linear number of processors on an EREW PRAM. Although the parallel algorithm follows the overall structure of our sequential algorithm, the parallelization of some of the steps required new insights into the problem. Our parallel algorithm can be made to run within the same time bound using a sublinear number of processors by using the algorithm for finding connected components in [3] and the algorithm for integer sorting in [9].

References

[1] G.-R. Cai and Y.-G. Sun, *The minimum augmentation of any graph to a k-edge-connected graph*, Networks, 19 (1989), pp. 151–172.

[2] R. Cole, *Parallel merge sort*, SIAM J. Comput., 17 (1988), pp. 770–785.

[3] R. Cole and U. Vishkin, *Approximate and exact parallel scheduling with applications to list, tree and graph problems*, in Proc. 27th Annual IEEE Symp. on Foundations of Comp. Sci., 1986, pp. 478–491.

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

[5] A. Frank, *Augmenting graphs to meet edge-connectivity requirements*, in Proc. 31th Annual IEEE Symp. on Foundations of Comp. Sci., 1990, pp. 708–718.

[6] H. Frank and W. Chou, *Connectivity considerations in the design of survivable networks*, IEEE Trans. on Circuit Theory, CT-17 (1970), pp. 486–490.

[7] G. N. Frederickson and J. Ja'Ja', *Approximation algorithms for several graph augmentation problems*, SIAM J. Comput., 10 (1981), pp. 270–283.

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

[9] T. Hagerup, *Towards optimal parallel bucket sorting*, Information and Computation, 75 (1987), pp. 39–51.

[10] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.

[11] 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.

[12] S. P. Jain and K. Gopal, *On network augmentation*, IEEE Trans. on Reliability, R-35 (1986), pp. 541–543.

[13] Y. Kajitani and S. Ueno, *The minimum augmentation of a directed tree to a k-edge-connected directed graph*, Networks, 16 (1986), pp. 181–197.

[14] R. M. Karp and V. Ramachandran, *Parallel algorithms for shared-memory machines*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., North Holland, 1990, pp. 869–941.

[15] D. Naor, D. Gusfield, and C. Martel, *A fast algorithm for optimally increasing the edge-connectivity*, in Proc. 31th Annual IEEE Symp. on Foundations of Comp. Sci., 1990, pp. 698–707.

[16] D. Nath and N. Maheshwari, *Parallel algorithms for the connected components and minimal spanning tree problems*, Information Processing Letters, 14 (1982), pp. 7–11.

[17] V. Ramachandran, *Parallel open ear decomposition with applications to graph biconnectivity and triconnectivity*, in Synthesis of Parallel Algorithms, J. H. Reif, ed., Morgan-Kaufmann, 1992, to appear.

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

[19] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, in Proc. 3rd Aegean Workshop on Computing, vol. LNCS #319, Springer-Verlag, 1988, pp. 111–123.

[20] D. Soroker, *Fast parallel strong orientation of mixed graphs and related augmentation problems*, Journal of Algorithms, 9 (1988), pp. 205–223.

[21] K. Steiglitz, P. Weiner, and D. J. Kleitman, *The design of minimum-cost survivable networks*, IEEE Trans. on Circuit Theory, CT-16 (1969), pp. 455–460.

[22] R. E. Tarjan, *Data Structures and Network Algorithms*, SIAM Press, Philadelphia, PA, 1983.

[23] R. E. Tarjan and U. Vishkin, *An efficient parallel biconnectivity algorithm*, SIAM J. Comput., 14 (1985), pp. 862–874.

[24] S. Ueno, Y. Kajitani, and H. Wada, *Minimum augmentation of a tree to a k-edge-connected graph*, Networks, 18 (1988), pp. 19–25.

[25] T. Watanabe, *An efficient way for edge-connectivity augmentation*, Tech. Rep. ACT-76-UILU-ENG-87-2221, Coordinated Science lab., University of Illinois, Urbana, IL, 1987.

[26] T. Watanabe and A. Nakamura, *On a smallest augmentation to triconnect a graph*, Tech. Rep. C-18, Department of Applied Mathematics, faculty of Engineering, Hiroshima University, Higashi-Hiroshima, 724, Japan, 1983. revised 1987.

[27] T. Watanabe and A. Nakamura, *Edge-connectivity augmentation problems*, J. Comp. System Sci., 35 (1987), pp. 96–144.

[28] T. Watanabe and A. Nakamura, 3-*connectivity augmentation problems*, in Proc. of 1988 IEEE Int'l Symp. on Circuits and Systems, 1988, pp. 1847–1850.

[29] T. Watanabe, T. Narita, and A. Nakamura, 3-*edge-connectivity augmentation problems*, in Proc. of 1989 IEEE Int'l Symp. on Circuits and Systems, 1989, pp. 335–338.