# On the Biconnected and Strongly Connected Components of the Complement of Graphs

Stavros D. Nikolopoulos and Leonidas Palios

*Department of Computer Science, University of Ioannina*
*P.O.Box 1186, GR-45110 Ioannina, Greece*
{stavros, palios}@cs.uoi.gr

**Abstract:** In this paper, we consider the problems of computing the biconnected components and the strongly connected components of the complement of a given graph. We describe simple sequential algorithms for these problems, which work on the input graph and not on its complement, and which for a graph on $n$ vertices and $m$ edges both run in optimal $O(n+m)$ time. Our algorithms are not data structure-based and they employ neither breadth-first-search nor depth-first-search.
Unlike previous linear co-biconnectivity and strong co-connectivity sequential algorithms, both algorithms admit efficient parallelization. The co-biconnectivity algorithm can be parallelized resulting in an optimal parallel algorithm that runs in $O(\log^2 n)$ time using $O((n+m)/\log^2 n)$ processors. Additionally, the strong co-connectivity algorithm can also be parallelized to yield an $O(\log^2 n)$-time and $O(m^{1.188}/\log n)$-processor solution. Moreover, the latter algorithm, implies a very simple optimal parallel co-connectivity algorithm.
Our results show that, in a parallel process environment, the problems of computing the biconnected components and the strongly connected components can be solved with better time-processor complexity on the complement of a graph rather than on the graph itself.

**Keywords:** Biconnected and co-biconnected components, strongly connected and co-connected components, co-biconnectivity algorithms, strong co-connectivity algorithms, parallel algorithms.

## 1 Introduction

A *biconnected component* of an undirected graph $G$ is a maximal set of edges of $G$ such that any two edges in the set lie on a common simple cycle [8]; the *biconnected co-components* of $G$ are the biconnected components of the complement $\overline{G}$ of $G$. A *strongly connected component* of a directed graph $G$ is a maximal set of vertices in which there is a directed path from each vertex to all the other vertices in the set; the *strongly connected co-components* of $G$ are the strongly connected components of the complement $\overline{G}$ of the directed graph $G$. From the definitions, it follows that the connected (biconnected) components of an undirected graph define a partition of the graph's vertices (edges, respectively), and the strongly connected components of a directed graph define a partition of the graph's vertices. These decompositions are fundamental tools in graph theory with applications in compiler analysis, data mining, scientific computing, and other areas. Thus, the computation of such components occupies a central place in algorithmic graph theory, both in a sequential and in a parallel process environment, and is a key step in algorithms for a number of combinatorial problems on graphs.

Sequentially, the problems of determining the biconnected and the strongly connected components of a graph are both solved by a search-and-label approach. For a graph on $n$ vertices and $m$ edges which is given in adjacency-list representation, simple sequential algorithms — e.g., based on depth-first search — for both problems run optimally in $O(n+m)$ time [8, 12].

By definition, the problems of determining the biconnected or the strongly connected co-components of a graph $G$ can be easily solved by first computing the complement $\overline{G}$ of $G$ and then by applying a biconnectivity or a strong connectivity algorithm on $\overline{G}$. It takes $\Omega(n^2)$ time to compute the complement explicitly, and thus, this approach leads to algorithms which may be super-linear in the size of the input graph. On the related problem of computing the connected co-components, Ito and Yokoyama [15] showed that a depth-first-search tree and a breadth-first-search tree on the complement of a given graph can be constructed in linear time; this result, in turn, implies a linear-time algorithm for computing the co-components of a graph. Dahlhaus, Gustedt, and McConnell, in their paper on modular decomposition [9], described a procedure for finding a depth-first-search forest on the complement of a directed graph in $O(n+m)$ time. In [10], the same authors gave simple algorithms for computing the connected and biconnected co-components of an undirected graph and the strongly connected co-components of a directed graph; this approach also relies on depth-first search. Recently, Chong, Nikolopoulos, and Palios [6] described a simple linear-time sequential algorithm for computing the connected co-components of a graph, which has the advantage of admitting efficient parallelization.

Developing efficient parallel algorithms for finding the biconnected and the strongly connected components and co-components of a graph turns out to be a more challenging problem. The problem is very important, especially for graphs with large size, for which parallel processing may be the only approach for obtaining a solution in reasonable time. Unfortunately, depth-first search (DFS) seems difficult to parallelize; indeed, Reif shows that a restricted version of the approach (lexicographic DFS) is $P$-complete [20].

Alternatively, there exist several parallel algorithms for the biconnected components problem and the strongly connected components problem that avoid the use of depth-first search. Early $O(\log n)$-time parallel biconnectivity algorithms appear in Tarjan and Vishkin [23], and Tsin and Chin [24]. These algorithms follow a strategy that is based on transforming the input graph $G$ into another graph $G'$ such than the biconnected components of $G$ are the connected components of $G'$; their starting point is an arbitrary spanning tree, rather than a depth-first-search tree. We note that the problem of computing the connected components of a graph has been extensively studied in the literature; we point out the recent work of Chong, Han, and Lam [5], which describes a parallel algorithm for computing the connected components of a graph in $O(\log n)$ time using $O(n+m)$ processors. For the problem of finding the strongly connected components of a graph, Gazit and Miller [11] proposed an NC algorithm, which is based upon matrix multiplication. This algorithm was improved by Cole and Vishkin [7], but still requires $O(\log^2 n)$ time and $O(n^{2.376})$ processors. Kao [17] developed a more complicated NC algorithm for planar graphs that requires $O(\log^3 n)$ time and $O(n/\log n)$ processors. An extensive coverage of parallel connectivity, biconnectivity and strong connectivity algorithms can be found in [1, 16, 19].

As in the sequential environment, the parallel computation of the biconnected and the strongly connected co-components of a graph can be easily done by computing the complement of the graph and then by applying one of the parallel algorithms for the biconnected and the strongly connected components on the complement. However, as in the sequential case, this yields non-optimal algorithms. To the best of our knowledge, no parallel algorithms which "directly" compute the biconnected and the strongly connected co-components exist. We mention that for the problem of computing the connected co-components of a graph on $n$ vertices and $m$ edges, the parallel version of the algorithm by Chong, Nikolopoulos, and Palios [6] runs in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM, and is therefore optimal.

In this paper, we establish properties of the complement of an undirected or a directed graph, which enable us to describe simple and efficient sequential algorithms for computing the biconnected and the strongly connected co-components of a graph without computing its complement. For a graph on $n$ vertices and $m$ edges, both the co-biconnectivity and the strong co-connectivity algorithm run in optimal $O(n+m)$ time. More interestingly, unlike previous such algorithms, both our algorithms admit efficient parallelization. In particular, the sequential co-biconnectivity algorithm leads to an optimal parallel algorithm that runs in $O(\log^2 n)$ time using $O((n+m)/\log^2 n)$ processors, while the strong co-connectivity algorithm gives an $O(\log^2 n)$-time and $O(m^{1.188}/\log n)$-processor parallel solution. As a byproduct of the latter algorithm, we obtain a very simple optimal parallel co-connectivity algorithm which is simpler than the one in [6]. Our proposed algorithms are not data structure-based and they employ neither breadth-first search nor depth-first search.

The paper is organized as follows. In Section 2 we present the notation and related terminology and we prove results on which the co-biconnectivity and the strong co-connectivity algorithms rely. In Section 3 we describe the sequential and the parallel co-biconnectivity algorithms, establish their correctness and analyze their complexity, while in Section 4 we describe the sequential and the parallel strong co-connectivity algorithms. Finally, in Section 5 we conclude the paper and present some open problems.

# 2 Theoretical Framework

We consider finite undirected graphs with no loops or multiple edges, and also directed graphs (by definition, directed graphs have no loops or multiple directed edges). Let $G$ be such a graph; then, $V(G)$ and $E(G)$ denote the set of vertices and of edges of $G$ respectively. An edge (undirected or directed) is a pair (undirected or directed) of distinct vertices $x, y \in V(G)$, and is denoted $xy$; if $G$ is an undirected graph, we say that $x$ is adjacent *to* $y$ (or $y$ is adjacent to $x$), whereas if $G$ is a directed graph we say that $x$ is adjacent *to* $y$ and $y$ is adjacent *from* $x$. For a set $V \subseteq V(G)$ of vertices of the graph $G$, the subgraph of $G$ *induced* by $V$ is denoted $G[V]$; for a set $\mathcal{E} \subseteq E(G)$ of edges, the subgraph of $G$ *spanned* by $\mathcal{E}$ is denoted $G\langle\mathcal{E}\rangle$.

The *neighborhood* $N(x)$ of a vertex $x \in V(G)$ of an undirected graph is the set of all the vertices of $G$ which are adjacent to $x$. The *closed neighborhood* of $x$ is defined as $N[x] := N(x) \cup \{x\}$. We use $M(x)$ to denote the set $V(G) - N[x]$. The *degree* of a vertex $x$ in an undirected graph $G$, denoted $d(x)$, is the number of vertices which are adjacent to $x$; thus, $d(x) = |N(x)|$. In the case where $G$ is a directed graph, $N^+(x)$ is the set of all the vertices of $G$ which are adjacent from $x$, and $N^-(x)$ is the set of all the vertices of $G$ which are adjacent to $x$. The *out-degree* of a vertex $x$ in $G$, denoted $d^+(x)$, is the number of vertices which are adjacent from $x$, and its *in-degree*, denoted $d^-(x)$, is the number of vertices which are adjacent to $x$; thus, $d^+(x) = |N^+(x)|$ and $d^-(x) = |N^-(x)|$.

A sequence of vertices $v_0 v_1 \cdots v_k$ is a *chain* of *length* $k$ in an arbitrary graph $G$ if $v_{i-1}v_i \in E(G)$ or $v_i v_{i-1} \in E(G)$ for $i = 1, 2, \ldots, k$. A sequence of vertices $v_0 v_1 \cdots v_k$ is a *path* from $v_0$ to $v_k$ of *length* $k$ in $G$ provided that $v_{i-1}v_i \in E(G)$ for $i = 1, 2, \ldots, k$. A *cycle* of length $k+1$ (or closed path) is a path $v_0 v_1 \cdots v_k$ such that $v_k v_0 \in E(G)$. A path (cycle) of an arbitrary graph $G$ is called *simple* if none of its vertices occurs more than once; it is called *trivial* if $k = 0$. An arbitrary graph $G$ is *connected* if between any two vertices there exists a chain in $G$ joining them; it is *strongly connected* if for any two vertices $x, y$ there exists a path in $G$ from $x$ to $y$.

A *connected component* (or component) of an undirected graph $G$ is a maximal set of vertices such that for every pair of vertices $x, y$ in the set, there exists a path from $x$ to $y$ in the subgraph of $G$ induced by the vertices in the set. The *co-connected components* (or co-components) of $G$ are the connected components of the complement $\overline{G}$ of $G$. A component (co-component) is called *non-trivial* if it contains two or more vertices; otherwise, it is called *trivial*.

A *biconnected component* (or bicomponent) of an undirected graph $G$ is a maximal set of edges such that any two edges in the set lie on a common simple cycle consisting of edges in the set [8]; the *biconnected co-components* (or co-bicomponents) of $G$ are the biconnected components of the complement $\overline{G}$ of $G$. The definition implies that each biconnected component contains at least three edges. For a set $\mathcal{E}$ of edges of the graph $G$, we say that $\mathcal{E}$ *has the biconnectivity property* if every pair of edges in $\mathcal{E}$ lie on a common simple cycle in the subgraph of $G$ spanned by the set $\mathcal{E}$. Clearly, all the edges of a set $\mathcal{E} \subseteq E(G)$ which has the biconnectivity property belong to the same biconnected component of $G$. A *cutpoint* (or *articulation point*) of a connected undirected graph $G$ is a vertex whose removal disconnects $G$, and a *bridge* is an edge with this property. By definition, a biconnected component contains no cutpoints or bridges.

A *strongly connected component* of a directed graph $G$ is a maximal set of vertices such that for any two vertices $x$ and $y$ in the set, there exists a directed path from $x, y$ in the subgraph of $G$ induced by the vertices in the set; the *strongly connected co-components* (or strongly co-components) of $G$ are the strongly connected components of the complement $\overline{G}$ of $G$.

Below we present some results on which our algorithms rely.

**Lemma 2.1.** [6] *Let $G$ be an undirected graph on $n$ vertices and $m$ edges. If $v$ is a vertex of $G$ of minimum degree, then the degree of $v$ does not exceed $\sqrt{2m}$.*

**Lemma 2.2.** *Let $G$ be a directed graph on $n$ vertices and $m$ edges. If $v$ is a vertex of $G$ of minimum sum of in-degree and out-degree, then the sum of in-degree and out-degree of $v$ does not exceed $2\sqrt{m}$.*

*Proof:* Let $d^-(x)$ and $d^+(x)$ denote the in-degree and the out-degree of vertex $x$ in $G$. Then, since $v$ is $G$'s vertex of minimum sum of in-degree and out-degree, we have that $\sum_x \big(d^-(x) + d^+(x)\big) \geq n \cdot \big(d^-(v) + d^+(v)\big) \Longrightarrow d^-(v) + d^+(v) \leq 2m/n$; then, if we take into account that $n > \sqrt{m}$ because $m \leq n \cdot (n-1) < n^2$, we get that $d^-(v) + d^+(v) \leq 2\sqrt{m}$. ∎

For an edge set $\mathcal{E}$ having the biconnectivity property, the following hold.

**Lemma 2.3.** *Let $G$ be an undirected graph, let set $\mathcal{E} \subseteq E(G)$ having the biconnectivity property, and let $V_\mathcal{E}$ be the set of vertices incident on the edges in $\mathcal{E}$. Then,*

*(i) for every edge $e \in \mathcal{E}$ and any two vertices $u, w \in V_\mathcal{E}$, the subgraph $G\langle\mathcal{E}\rangle$ of $G$ spanned by the edges in $\mathcal{E}$ contains a simple path from $x$ to $y$ that passes through $e$;*

*(ii) for any simple path in $G$ with endpoints in $V_\mathcal{E}$, the set $\mathcal{E} \cup \mathcal{P}$, where $\mathcal{P}$ is the edge set of the path, has the biconnectivity property;*

*(iii) the edge set of the subgraph of $G$ induced by $V_\mathcal{E}$ has the biconnectivity property;*

*(iv) for any set $\mathcal{E}' \subseteq E(G)$ such that $\mathcal{E}'$ has the biconnectivity property and $\mathcal{E} \cap \mathcal{E}' \neq \emptyset$, the set $\mathcal{E} \cup \mathcal{E}'$ also has the biconnectivity property.*

*Proof:* (i) Since the set $\mathcal{E}$ has the biconnectivity property, there exists a simple path, say, $\rho$, in the graph $G\langle\mathcal{E}\rangle$ from $u$ to $w$. Additionally, if $e'$ is any edge of $\rho$, there exists a simple cycle in $G\langle\mathcal{E}\rangle$ that passes through both $e$ and $e'$. If the edge $e$ is removed, this simple cycle becomes a simple path $\rho' = a \cdots b$, where $a, b$ are the endpoints of $e$. Let $x$ ($y$ respectively) be the first (last respectively) common vertex of the paths $\rho$ and $\rho'$ as we move from $a$ to $b$ along $\rho'$; because the two paths share the edge $e'$, the vertices $x, y$ are well defined and distinct. Then, if $\rho = u \cdots x \cdots y \cdots w$, the path that consists of the part of $\rho$ from $u$ to $x$, followed by the part of $\rho'$ from $x$ to $a$, followed by the edge $e$, followed by the part of $\rho'$ from $b$ to $y$, followed by the part of $\rho$ from $y$ to $w$ is a simple path from $u$ to $w$ through $e$, as desired. Otherwise, $\rho = u \cdots y \cdots x \cdots w$, and the desired path is formed by the part of $\rho$ from $u$ to $y$, followed by the part of $\rho'$ from $y$ to $b$, followed by the edge $e$, followed by the part of $\rho'$ from $a$ to $x$, followed by the part of $\rho$ from $x$ to $w$.

(ii) Let the simple path be $\rho$ and let $u, w \in V_{\mathcal{E}}$ be its endpoints. First, let us assume that none of the vertices of $\rho$ except for its endpoints $u, w$ belong to $V_{\mathcal{E}}$. We show that, for any pair of edges in $\mathcal{E} \cup \mathcal{P}$, the two edges lie on a common simple cycle in the subgraph $G\langle \mathcal{E} \cup \mathcal{P} \rangle$ of $G$ spanned by the edges in $\mathcal{E} \cup \mathcal{P}$. This clearly holds for any two edges in $\mathcal{E}$. Next, consider the case where $e \in \mathcal{E}$ and $e' \in \mathcal{P}$: since, according to Lemma 2.3(i), there exists a simple path along edges in $\mathcal{E}$ from $u$ to $w$ that passes through $e$, then the concatenation of this path with $\rho$ gives a simple cycle in $G\langle \mathcal{E} \cup \mathcal{P} \rangle$ that passes through both $e$ and $e'$. Finally, consider the case where $e, e' \in \mathcal{P}$: then, any simple path along edges in $\mathcal{E}$ from $u$ to $w$ concatenated with $\rho$ gives the desired simple cycle.

Suppose now that there exist vertices of the path $\rho$, other than its endpoints, that belong to $V_{\mathcal{E}}$. Let us remove from $\rho$ any edges in $\mathcal{E}$ and let us break the resulting sub-paths at their vertices that belong to $V_{\mathcal{E}}$; we obtain a collection of simple paths $\rho_1, \rho_2, \ldots, \rho_k$. Then, as shown earlier, the union of $\mathcal{E}$ and the edge set of $\rho_1$ has the biconnectivity property; in turn, the union of this set with the edge set of $\rho_2$ also has the biconnectivity property, and so on so forth; eventually, after having considered all the paths $\rho_1, \rho_2, \ldots, \rho_k$, we have that the set $\mathcal{E} \cup \mathcal{P}$ has the biconnectivity property.

(i) Trivially true due to Lemma 2.3(ii), since any edge $e \notin \mathcal{E}$ whose endpoints belong to $V_{\mathcal{E}}$ constitutes a simple path connecting two vertices in $V_{\mathcal{E}}$.

(iv) Follows directly from Lemma 2.3(ii); note that if $e = uw$ is an edge in $\mathcal{E} \cap \mathcal{E}'$, then for any edge $e' \in \mathcal{E}' - \mathcal{E}$, the biconnectivity of $\mathcal{E}'$ implies that there exists a simple cycle that passes through $e$ and $e'$, or equivalently that there exists a simple path of edges in $\mathcal{E}'$ from $u \in V_{\mathcal{E}}$ to $w \in V_{\mathcal{E}}$ through edge $e'$. ∎

This lemma has the following interesting implications.

**Lemma 2.4.** *Let $G$ be an undirected graph, let $\mathcal{E}_1, \mathcal{E}_2 \subseteq E(G)$ be disjoint sets of edges having the biconnectivity property, and let $V_1$ and $V_2$ be the sets of vertices incident on the edges in $\mathcal{E}_1$ and $\mathcal{E}_2$ respectively.*

*(i) If $V_1 \cap V_2 = \emptyset$ and there exist distinct vertices $u, w \in V_1$ and $x, y \in V_2$ such that $ux \in E(G)$ and $wy \in E(G)$, then the edge set of the subgraph of $G$ induced by $V_1 \cup V_2$ has the biconnectivity property.*

*(ii) Suppose that $V_1 \cap V_2 = \{v\}$. If there exists a simple path $\rho$ from a vertex in $V_1$ to a vertex in $V_2$ which does not go through vertex $v$, then the edge set of the subgraph of $G$ induced by $V_1 \cup V_2 \cup V(\rho)$ has the biconnectivity property.*

*(iii) If $|V_1 \cap V_2| \geq 2$, then the edge set of the subgraph of $G$ induced by $V_1 \cup V_2$ has the biconnectivity property.*

*Proof:* (i) In light of Lemma 2.3(iii), it suffices to show that the set $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \{ux, wy\}$ has the biconnectivity property. The biconnectivity of $\mathcal{E}_1$ and $\mathcal{E}_2$ implies that for any two edges in $\mathcal{E}_1$ or $\mathcal{E}_2$, there exists a simple cycle in the subgraph $G\langle \mathcal{E} \rangle$ spanned by the edges in $\mathcal{E}$ which passes through them. Additionally, for any edge $e_1 \in \mathcal{E}_1$ and any edge $e_2 \in \mathcal{E}_2$, Lemma 2.3(i) implies that there exist paths of edges in $\mathcal{E}_1$ and $\mathcal{E}_2$ that lead from $u$ to $w$ through $e_1$, and from $x$ to $y$ through $e_2$ respectively; these paths and the edges $ux$ and $wy$ form a simple cycle in $G\langle \mathcal{E} \rangle$ through $e_1$ and $e_2$. Since this very cycle also goes through the edges $ux$ and $wy$, it helps resolve the cases of pairs of edges $ux$ and $e$, and $wy$ and $e$, for any edge $e \in \mathcal{E}_1 \cup \mathcal{E}_2$.

(ii) It suffices to show that the set $\mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{P}$ has the biconnectivity property, where $\mathcal{P}$ is the edge set of the path $\rho$. Let $u \in V_1$ and $x \in V_2$ be the endpoints of $\rho$, and let $y$ be the first vertex in $V_2$ that we meet as we move along $\rho$ from $u$ to $x$; we break the path $\rho$ at $y$ and we obtain two simple paths $\rho_1$ (with endpoints $u$ and $y$) and $\rho_2$ (with endpoints $x$ and $y$). The biconnectivity of $\mathcal{E}_2$ implies that there exists a simple path from $y$ to $v$ along edges in $\mathcal{E}_2$. By concatenating $\rho_1$ and this path we obtain a simple path $\rho'$ whose both endpoints belong to $V_1$; then, Lemma 2.3(ii)

implies that the union of the edge set $E(\rho')$ of $\rho'$ and $\mathcal{E}_1$ has the biconnectivity property. In turn, this implies that the set $\mathcal{E}_1 \cup \mathcal{E}_2 \cup E(\rho') = \mathcal{E}_1 \cup \mathcal{E}_2 \cup E(\rho_1)$ has the biconnectivity property since the sets $\mathcal{E}_2$ and $E(\rho')$ share edges (Lemma 2.3(iv)). Finally, since the endpoints of the path $\rho_2$ belong to the vertex set of $\mathcal{E}_1 \cup \mathcal{E}_2 \cup E(\rho_1)$, its edge set can also be included and the resulting set has the biconnectivity property (Lemma 2.3(ii)).
(iii) It suffices to show that the set $\mathcal{E}_1 \cup \mathcal{E}_2$ has the biconnectivity property. Let $x, y \in V_1 \cap V_2$. Since the set $\mathcal{E}_1$ has the biconnectivity property, there exists a simple path $\rho$ of edges in $\mathcal{E}_1$ from $x$ to $y$ (Lemma 2.3(i)). Then, Lemma 2.3(ii) implies that the union $\mathcal{A} = \mathcal{E}_2 \cup \mathcal{P}$, where $\mathcal{P}$ is the edge set of the path $\rho$, has the biconnectivity property; finally, Lemma 2.3(iv) completes the proof since the sets $\mathcal{A}$ and $\mathcal{E}_1$ have the biconnectivity property and they are not disjoint. ∎

Observe that Lemma 2.4(iii) implies that if two or more vertices belong to the vertex set of a biconnected component, they belong together to exactly one such set.

## 3 Biconnected Components of the Complement of a Graph

Let $G$ be an undirected graph on $n$ vertices and $m$ edges. We next present an $O(n+m)$-time algorithm for computing the biconnected components of the complement $\overline{G}$ of $G$, which can be parallelized resulting in an optimal algorithm that runs in $O(\log^2 n)$ time using $O((n+m)/\log^2 n)$ processors on the CREW PRAM model of computation. The algorithm relies on Lemma 2.1 and the result established in the following lemma.

**Lemma 3.1.** *Let $G$ be an undirected graph on $m \geq 1$ edges and $x$ be a vertex of $G$. If $C_1, C_2, \ldots, C_k$ are the connected components of the subgraph of the complement $\overline{G}$ induced by the set $M(x)$ of non-neighbors of $x$ in $G$, then*

- *(i) the vertex sets $C_1, C_2, \ldots, C_k$ are disjoint;*
- *(ii) their number $k$ does not exceed $2\sqrt{m}$;*
- *(iii) for each $C_i$ of cardinality at least 2, the edge set of the subgraph of the complement $\overline{G}$ induced by $C_i \cup \{x\}$ has the biconnectivity property.*

*Proof:* (i) Follows from the definition of connected components.
(ii) If $k = 1$, the lemma clearly holds, since $m \geq 1$. Suppose now that $k \geq 2$. Since $C_1, C_2, \ldots, C_k$ are the connected components of $\overline{G}[M(x)]$, then every vertex in $C_i$ is adjacent in $G$ to every vertex in $C_j$, for all $j \neq i$. Thus, $G$ contains at least $k(k-1)/2$ edges, that is, $k(k-1)/2 \leq m$. The lemma follows since for $k \geq 2$ we have that $k(k-1)/2 \geq k^2/4$.
(iii) It is trivial to see that, for any edge $e = yz$ of the subgraph $\overline{G}[C_i]$ of $\overline{G}$ induced by the vertices in $C_i$, the edge set $\{xy, xz, yz\}$ has the biconnectivity property. Then, consider the following process: we start with an arbitrary edge $yz$ of $\overline{G}[C_i]$ and we initialize the current edge set having the biconnectivity property to the set $\{xy, xz, yz\}$; then, we traverse $\overline{G}[C_i]$ in, say, a depth-first-search manner, and for each edge $y'z'$ we traverse, we insert the edges $xy', xz', y'z'$ in the current edge set; in each step, the resulting edge set is guaranteed to have the biconnectivity property due to Lemma 2.4(iii), since the vertex sets of the two merged edge sets share at least two elements. The lemma follows from the fact that the graph $\overline{G}[C_i]$ is connected. ∎

The algorithm receives as input an undirected graph given in adjacency-list representation. It returns the biconnected components of the graph $\overline{G}$ as follows: for each such biconnected component, it returns a subset $\mathcal{B}_i$ of its edges such that every vertex of the component is incident on at least one edge in $\mathcal{B}_i$. In other words, each biconnected component is the subgraph of $\overline{G}$ induced by the vertices incident on the edges in one of the sets $\mathcal{B}_i$. (It is important to note that returning the edge sets of the biconnected components of $\overline{G}$ could require $\Theta(n^2)$ time, where $n$ is the number of vertices of the graph $G$, since the sum of sizes of all the edge sets could be equal to $|E(\overline{G})|$ which may be as large as $\Theta(n^2)$.) The total description size of the sets $\mathcal{B}_i$ is linear in the size of the input graph.

Algorithm BICONNECTED_CO-COMPONENTS
*for the computation of the biconnected components of the complement of a graph*

*Input:* an undirected graph $G$.

*Output:* the biconnected components of the complement $\overline{G}$.

1. $v \leftarrow$ a vertex of minimum degree in $G$;
   let $N(v)$ and $M(v)$ be the sets of neighbors and non-neighbors of $v$ in $G$;
   $S \leftarrow N(v) \cup \{v\}$; {*$S$ will be a superset of the set of cutpoints and bridge endpoints in $\overline{G}$*}

2. if the degree of $v$ in the graph $G$ is 0
   then compute the co-components $C_1, C_2, \ldots, C_k$ of $G[V(G) - \{v\}]$;
   the biconnected components of $\overline{G}$ are the edge sets of the subgraphs $\overline{G}[C_i \cup \{v\}]$, for the non-trivial co-components $C_i$ (i.e., $|C_i| \geq 2$);
   exit;

3. $R \leftarrow \emptyset$;
   for each vertex $u$ in $N(v)$ do
   compute the number $\mu_u$ of neighbors of $u$ (in $G$) which belong to $M(v)$;
   if $\mu_u = |M(v)| - 1$ or $\mu_u = |M(v)|$
   then insert the vertex $u$ in $R$;
   if $\mu_u = |M(v)| - 1$ and $w$ is the non-neighbor of $u$ in $G$ which belongs to $M(v)$
   then insert the vertex $w$ in $S$; {*cutpoint*}

4. construct the graph $G[M(v)]$ and compute its co-components $C_1, C_2, \ldots, C_k$;
   for each co-component $C_i$, $i = 1, 2, \ldots, k$, do
   if $C_i$ is a trivial co-component (i.e., $|C_i| = 1$)
   then insert the only element of $C_i$ in $S$; {*potential bridge endpoint*}

5. construct the auxiliary graph $\widehat{G}$ with vertex and edge sets as follows:
   $V(\widehat{G}) = S \cup \{\, p_i, q_i \mid \forall \text{ non-trivial } C_i \,\}$
   $E(\widehat{G}) = E_1 \cup E_2 \cup E_3 \cup E_4$
   where
   $E_1 = \{\, xy \mid x, y \in S \text{ and } xy \notin E(G) \,\}$,
   $E_2 = \{\, p_i q_i, p_i v, q_i v \mid \forall \text{ non-trivial } C_i \,\}$,
   $E_3 = \{\, q_i w \mid \forall \text{ non-trivial } C_i \text{ and } w \in C_i \cap S \,\}$,
   $E_4 = \{\, up_i, uq_i \mid \forall \text{ non-trivial } C_i, u \in N(v) - R, \text{ and } \exists\, y \in C_i : uy \notin E(G) \,\}$;

6. compute the biconnected components (edge sets) $\widehat{\mathcal{B}}_1, \widehat{\mathcal{B}}_2, \ldots, \widehat{\mathcal{B}}_\ell$ of $\widehat{G}$;
   for each $i = 1, 2, \ldots, \ell$ do
   $\mathcal{B}_i \leftarrow \widehat{\mathcal{B}}_i$;
   for each co-component $C_i$ of $G[M(v)]$, $i = 1, 2, \ldots, k$, do
   if $C_i$ is non-trivial and the edge $p_i q_i$ belongs to the biconnected component $\widehat{\mathcal{B}}_j$
   then for each vertex $w \in C_i - S$ do
   insert the edge $vw$ in $\mathcal{B}_j$;
   remove from $\mathcal{B}_j$ all the edges incident on $p_i$ or $q_i$;

7. print the resulting edge sets $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_\ell$; the biconnected components of $\overline{G}$ are the edge sets of the subgraphs of $\overline{G}$ induced by the vertices of the subgraphs spanned by $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_\ell$;

We note that the set $R$ contains the vertices in $N(v)$ which have at most one non-neighbor in $G$ which belongs to $M(v)$. Moreover, if a vertex $u \in N(v)$ has exactly one non-neighbor, say, $w$, belonging to $M(v)$, then the graph $\widehat{G}$ contains both $u$ and $w$ in its vertex set, and the edge $uw$ in its edge set; the vertex $u$ is added in $S$ in Step 1 because $u \in N(v)$, and the vertex $w$ in Step 3.

We also note that the graph $\widehat{G}$ models the complement $\overline{G}$ of $G$, where the sets of vertices $C_i - S$, for each non-trivial co-component $C_i$ (i.e., $|C_i| \geq 2$), have been replaced by the vertices $p_i, q_i$. Additionally, the neighborhood of the vertex $v$ in $\widehat{G}$ includes the vertices in $C_i \cap S$, for all the co-components $C_i$ (see Step 4), and the vertices $p_i, q_i$, for each non-trivial co-component $C_i$. Since, for each non-trivial $C_i$, the vertices $p_i, q_i$, and the vertices in $C_i \cap S$ induce a connected subgraph, we have the following observation.

**Observation 3.1** *Let $G$ be an undirected graph, $v$ a vertex of $G$, and let the co-components $C_1$, $C_2, \ldots, C_k$ and the graph $\widehat{G}$ be as defined in Steps 4 and 5 of the algorithm. The components of the subgraph of $\widehat{G}$ induced by the neighbors of $v$ are*

- *the sets $\{p_i, q_i\} \cup (C_i \cap S)$, for each non-trivial co-component $C_i$, and*
- *the trivial co-components $C_j$.*

Before proving the correctness of the algorithm, we state and prove a few important facts about the set $S$ and the auxiliary graphs involved in the algorithm.

**Lemma 3.2.** *Let $G$ be an undirected graph on which we apply the algorithm* BICONNECTED_CO-COMPONENTS*, and let $S$ be the set of vertices collected in Steps 1, 3, and 4 of the algorithm. Then, the set $S$ contains all the cutpoints and the endpoints of all the bridges of the complement $\overline{G}$ of $G$.*

*Proof:* Let $x$ be a cutpoint or an endpoint of a bridge in $\overline{G}$. The definition of a cutpoint or a bridge implies that either $x$ has degree 1 in $\overline{G}$ or its removal disconnects a connected component of $\overline{G}$. Since $N[v] \subseteq S$ (see Step 1), where $N[v]$ is the closed neighborhood of $v$ in $G$, we need consider only the case that $x \in M(v)$, where $M(v)$ is the set of non-neighbors of $v$ in $G$. Then, the degree of $x$ in $\overline{G}$ is at least equal to 1, since $vx \in E(\overline{G})$. If the degree is 1, then $x$'s only neighbor in $\overline{G}$ is $v$, and thus $x$ forms a single-vertex connected component of $\overline{G}[M(v)]$. But then $x$ is inserted in the set $S$ in Step 4.

Suppose now that the vertex $x$ belongs to a connected component $P$ of $\overline{G}$ and that $x$'s removal disconnects the subgraph $\overline{G}[P]$. Then, $M(v) \cup \{v\} \subseteq P$, as all the vertices in $M(v)$ are connected in $\overline{G}$ through $v$. In fact, $P$ contains vertices from $N(v)$ as well, for otherwise the removal of $x$ would not disconnect $\overline{G}[P]$. When $x$ is removed, the vertices in $M(v) \cup \{v\}$ (and perhaps some vertices in $N(v)$ as well) remain connected, yet there exists a subset $P'$ of $P$ which are no longer connected in $\overline{G}$ to the vertices in $M(v) \cup \{v\}$. Hence, $P' \subseteq N(v)$ and every vertex in $M(v) - \{x\}$ is adjacent to every vertex in $P'$. But then, if $u$ is a vertex in $P'$, the number of neighbors of $u$ (in $G$) which belong to $M(v)$ is $|M(v)| - 1$, and $x$ is $u$'s only non-neighbor in $M(v)$; thus, $x$ is inserted in the set $S$ in Step 3. ∎

**Lemma 3.3.** *Let $G$ be an undirected graph, $v$ a vertex of $G$, and let the graph $\widehat{G}$ and its biconnected components $\widehat{\mathcal{B}}_1, \widehat{\mathcal{B}}_2, \ldots, \widehat{\mathcal{B}}_\ell$ be as defined in Steps 5 and 6. Then, for each non-trivial co-component $C_i$ of the graph $G[M(v)]$, the vertices $p_i, q_i$ belong to exactly one and the same biconnected component of $\widehat{G}$, which also contains $v$ and the vertices in $C_i \cap S$.*

*Proof:* First, observe that the edge set of the subgraph of $\widehat{G}$ induced by the vertices in $\{p_i, q_i, v\} \cup (C_i \cap S)$ has the biconnectivity property; note that $\widehat{G}$ contains the cycle $p_i q_i v$, as well as the cycle $w q_i v$, for each vertex $w \in C_i \cap S$. Thus, there exists a biconnected component of $\widehat{G}$ whose vertex set contains $p_i, q_i$, and the vertices in $C_i \cap S$. Additionally, any such component $\widehat{\mathcal{B}}_j$ containing $p_i$ also contains $q_i$: Since $p_i$ belongs to the vertex set of $\widehat{\mathcal{B}}_j$, there exists a simple cycle $O$ along edges in $\widehat{\mathcal{B}}_j$ which passes from $p_i$. If the cycle $O$ goes through $q_i$, then what we claimed holds; if it does not go through $q_i$, then because in $\widehat{G}$ the vertex $q_i$ is adjacent to all the neighbors of $p_i$ (except for $q_i$), we can replace $p_i$ in $O$ by $q_i$ and get a simple cycle that shares edges with $O$, which implies again that $q_i$ belongs to the vertex set of $\widehat{\mathcal{B}}_j$ (Lemma 2.3(iv)).

Finally, any biconnected component of $\widehat{G}$ containing $q_i$ also contains $p_i$: As above, there exists a simple cycle $O'$ along edges of the component which passes from $q_i$. If $O'$ goes through $p_i$ as well, we are done; if not, then if it goes along an edge $q_i w$, where $w \in C_i \cap S$, again the component contains $p_i$ as well, because its vertex set shares two vertices with the vertices in $\{p_i, q_i, v\} \cup (C_i \cap S)$ (Lemma 2.4(iii)), otherwise, the neighbors of $q_i$ in the cycle $O'$ are also neighbors of $p_i$ (since they belong to $N(v) \cup \{v\}$) and thus we can replace $q_i$ in $O'$ by $p_i$ and get a simple cycle that shares edges with $O'$, which implies again that $p_i$ belongs to the vertex set of the biconnected component. ∎

The correctness of the algorithm follows from the correctness of Steps 2 and 6. The correctness of Step 2 is established in Lemma 3.4; the correctness of Step 6 follows from the fact that all the edges in the graph $\widehat{G}$ which are not incident on any $p_i$ or $q_i$ are edges of the complement $\overline{G}$, from Lemma 3.3, and from Lemma 3.6 with the aid of Lemma 3.5.

**Lemma 3.4.** *Let $G$ be an undirected graph, $v$ be a vertex of $G$ whose degree is 0, and let $C_1, C_2, \ldots, C_k$ be the connected components of the subgraph $\overline{G}[V(G) - \{v\}]$. Then, the biconnected components of the complement $\overline{G}$ of $G$ are the edge sets of the subgraphs of $\overline{G}$ induced by the vertex sets $C_i \cup \{v\}$, for each non-trivial component $C_i$.*

*Proof:* Clearly, the vertices of the graph $G$ other than $v$ are partitioned in the sets $C_1, C_2, \ldots, C_k$. For each singleton set $C_i$, the unique vertex in $C_i$ is adjacent in the complement $\overline{G}$ only to $v$; thus, the edge connecting it to $v$ is a bridge in $\overline{G}$ and does not belong to any biconnected component of $\overline{G}$. For any other set $C_j$, we have that $|C_j| \geq 2$, which implies that the edge set of the subgraph $\overline{G}[C_j \cup \{v\}]$ has the biconnectivity property (see the proof of Lemma 3.1(iii)). The lemma follows from the fact that the edge set of $\overline{G}$ contains precisely the bridges $vw$ (for all vertices $w$ belonging to trivial components $C_i$) and the edges of the subgraphs $\overline{G}[C_j \cup \{v\}]$ (for the non-trivial co-components $C_j$), and from the observation that the subgraphs $\overline{G}[C_j \cup \{v\}]$ and $\overline{G}[C_{j'} \cup \{v\}]$, for any two non-trivial co-components $C_j$ and $C_{j'}$, share only vertex $v$, and thus their edge sets do not belong to the same biconnected component of $\overline{G}$. ∎

**Lemma 3.5.** *Let $G$ be an undirected graph, $v$ a vertex of $G$, and $C_1, C_2, \ldots, C_k$ the co-components of the subgraph $G[M(v)]$. Then,*

*(i) Let $u$ be a neighbor of $v$ in $G$. If $u$ is non-adjacent to vertices $w, w' \in M(v)$ and $w$ belongs to the co-component $C_i$, then the edges $uw, uw', vw, vw'$ of the complement $\overline{G}$ as well as all the edges of the subgraph $\overline{G}[C_i]$ belong to the same biconnected component of $\overline{G}$.*

*(ii) Let $u, u'$ be two neighbors of $v$ in $G$ such that in $G$ vertex $u$ is non-adjacent to vertices $w_1, w_2 \in M(v)$, vertex $u'$ is non-adjacent to vertices $w'_1, w'_2 \in M(v)$, and $w_1, w'_1$ belong to the same co-component $C_i$. Then, the edges $uw_1, uw_2, u'w'_1, u'w'_2, vw_1, vw_2, vw'_1, vw'_2$ of the complement $\overline{G}$ as well as all the edges of the subgraph $\overline{G}[C_i]$ belong to the same biconnected component of $\overline{G}$.*

*Proof:* (i) Clearly, the edges $\mathcal{E} = \{uw, uw', vw, vw'\}$ of $\overline{G}$ have the biconnectivity property. If the co-component $C_i$ is trivial (i.e., $|C_i| = 1$), then $C_i = \{w\}$ and the subgraph $\overline{G}[C_i]$ has no edges; hence, the edges $uw, uw', vw, vw'$ and the empty set of edges of $\overline{G}[C_i]$ belong to the same biconnected component of $\overline{G}$. If $C_i$ is non-trivial, Lemma 3.1(iii) implies that the edges of the subgraph $\overline{G}[C_i]$ as well as the edges of $\overline{G}$ connecting $v$ with all the vertices in $C_i$ belong to the same biconnected component of $\overline{G}$; the stated claim follows from the fact that this edge set and $\mathcal{E}$ share the edge $vw$ (see Lemma 2.4(iii)).

(ii) Statement (i) implies that the edges of $\overline{G}$ in $\mathcal{E} = \{uw_1, uw_2, vw_1, vw_2\} \cup E(\overline{G}[C_i])$ belong to the same biconnected component of $\overline{G}$. A similar fact holds for the set $\mathcal{E}' = \{u'w'_1, u'w'_2, vw'_1, vw'_2\} \cup E(\overline{G}[C_i])$ associated with the vertex $u'$ and the vertices $w'_1, w'_2$. Then, if the co-component $C_i$ is trivial, we have that $w_1 = w'_1$ and the two edge sets $\mathcal{E}$ and $\mathcal{E}'$ share edge $vw_1$; if $C_i$ is non-trivial, then the subgraph $\overline{G}[C_i]$ has at least one edge, which is shared by $\mathcal{E}$ and $\mathcal{E}'$. In either case, due to Lemma 2.3(iv), all the edges belong to the same biconnected component of $\overline{G}$. ∎

**Lemma 3.6.** *The algorithm* BICONNECTED_CO-COMPONENTS *correctly computes the biconnected components of the complement $\overline{G}$ of the input graph $G$.*

*Proof:* Let $R$ and $S$, and $\widehat{G}$ be the sets and the auxiliary graph computed by the algorithm BICONNECTED_CO-COMPONENTS when applied on $G$. In light of Lemma 2.4(iii), the proof relies on showing that any two vertices $x_1, x_2$ belong to the same biconnected component of $\overline{G}$ if and only if $x_1, x_2 \in V(\mathcal{B}_t)$ for some $t \in \{1, 2, \ldots, \ell\}$.

($\Longrightarrow$) Suppose that the vertices $x_1, x_2$ belong to the same biconnected component of $\overline{G}$. If $x_1, x_2$ belong to the same non-trivial co-component of $G[M(v)]$, then $x_1, x_2$ belong to the vertex set of the same set $\mathcal{B}_t$ due to Lemma 3.3 and Step 6 of the algorithm BICONNECTED_CO-COMPONENTS. Thus, suppose that not both $x_1, x_2$ belong to the same non-trivial co-component of $G[M(v)]$. In order to show that $x_1, x_2$ belong to the vertex set of the same set $\mathcal{B}_t$, due to Step 6, it suffices to show that $\widehat{x}_1, \widehat{x}_2$ belong to the same biconnected component of the graph $\widehat{G}$, where for $r = 1, 2$, $\widehat{x}_r = q_i$ if $x_r$ belongs to the non-trivial co-component $C_i$ of $G[M(v)]$, and $\widehat{x}_r = x_r$ otherwise; note that since $x_1, x_2$ do not belong to the same co-component of $G[M(v)]$, then $\widehat{x}_1 \neq \widehat{x}_2$.

Since the vertices $x_1, x_2$ belong to the same biconnected component of $\overline{G}$, $\overline{G}$ contains a simple cycle $O$ which passes through $x_1$ and $x_2$. We will replace parts of the cycle $O$ by paths in the graph $\widehat{G}$, obtaining a simple cycle in $\widehat{G}$ which passes through $\widehat{x}_1$ and $\widehat{x}_2$. Since all the vertices in $N(v) \cup \{v\}$ as well as those inducing trivial co-components of $G[M(v)]$ belong to the set $S$, they appear in $\widehat{G}$; thus, we need only concentrate in the parts of the cycle $O$ whose vertices belong to the non-trivial co-components of $G[M(v)]$. So, for each non-trivial co-component $C_i$ of $G[M(v)]$ which contributes vertices to the cycle $O$, we do the following:

(i) *$x_1 \in C_i$ and $x_2 \notin C_i$:* Then, let $aa' \cdots x_2 \cdots b'b$ be the subpath of the cycle $O$ passing through $x_2$ such that only $a, b$ belong to $C_i$. Then, we replace the part $a'a \cdots x_1 \cdots bb'$ by
  the path $a'aq_ibb'$ if $a, b \in C_i \cap S$,
  the path $a'aq_ib'$ if $a \in C_i \cap S$ and $b \in C_i - S$,
  the path $a'q_ibb'$ if $a \in C_i - S$ and $b \in C_i \cap S$, and
  the path $a'p_iq_ib'$ if $a, b \in C_i - S$.
In each case, the vertex $x_1$ has been replaced by the vertex $q_i$, which is precisely $\widehat{x}_1$.

(ii) *$x_2 \in C_i$ and $x_1 \notin C_i$:* This case is symmetric to case (i).

(iii) *$x_1, x_2 \notin C_i$:* Let $aa' \cdots x_1 \cdots b'b$ and $cc' \cdots x_2 \cdots d'd$ be the subpaths of the cycle $O$ such that none of the vertices except for $a, b, c, d$ belongs to $C_i$ (note that these two paths can be linked to form a cycle by connecting either $a$ to $d$ and $b$ to $c$, or $a$ to $c$ and $b$ to $d$; this allows us to ignore several symmetric cases in our analysis). Clearly, $a \neq b$ and $c \neq d$. We distinguish the following cases:

- $a = d$ and $b = c$: Then, if $a \in C_i - S$, we replace $a'ad'$ by $a'p_id'$, otherwise we leave it as it is; if $b \in C_i - S$, we replace $b'bc'$ by $b'q_ic'$, otherwise we leave it as it is.
- $a = d$ and $b \neq c$: Then, if $a \in C_i - S$, we replace $a'ad'$ by $a'p_id'$, otherwise we leave it as it is; the path $b'b \cdots cc'$ is replaced by
  $b'bq_icc'$ if $b, c \in C_i \cap S$,
  $b'bq_ic'$ if $b \in C_i \cap S$ and $c \in C_i - S$,
  $b'q_icc'$ if $b \in C_i - S$ and $c \in C_i \cap S$, and
  $b'q_ic'$ if $b, c \in C_i - S$.

- $a, b, c, d$ are all distinct: Suppose first that the vertex $v$ does not belong to any of the paths $aa' \cdots x_1 y_1 \cdots b'b$ and $cc' \cdots x_2 y_2 \cdots d'd$. Then, we replace the paths $a'a \cdots dd'$ and $b'b \cdots cc'$ by
  $a'avdd'$ and $b'bq_icc'$ if $a, b, c, d \in C_i \cap S$,
  $a'avp_id'$ and $b'bq_icc'$ if $a, b, c \in C_i \cap S$ and $d \in C_i - S$,
  $a'avp_id'$ and $b'bq_ic'$ if $a, b \in C_i \cap S$ and $c, d \in C_i - S$,
  $a'avp_id'$ and $b'q_icc'$ if $a, c \in C_i \cap S$ and $b, d \in C_i - S$,
  $a'aq_id'$ and $b'p_ic'$ if $a \in C_i \cap S$ and $b, c, d \in C_i - S$, and
  $a'p_id'$ and $b'q_ic'$ if $a, b, c, d \in C_i - S$.
  Let us now consider the case where $v$ appears in (one of) the paths; let it be in $cc' \cdots x_2 y_2 \cdots v \cdots d'd$. Then, we replace the paths $a'a \cdots dd' \cdots v$ and $b'b \cdots cc'$ by
  $a'av$ and $b'bq_icc'$ if $a, b, c \in C_i \cap S$,
  $a'av$ and $b'bq_ic'$ if $a, b \in C_i \cap S$ and $c \in C_i - S$,
  $a'av$ and $b'q_ic'$ if $a, b \in C_i \cap S$ and $c \in C_i - S$,
  $a'av$ and $b'bq_ic'$ if $a, c \in C_i \cap S$ and $b \in C_i - S$,
  $a'av$ and $b'q_ic'$ if $a \in C_i \cap S$ and $b, c \in C_i - S$,
  $a'p_iv$ and $b'q_icc'$ if $c \in C_i \cap S$ and $a, b \in C_i - S$, and
  $a'p_iv$ and $b'q_ic'$ if $a, b, c \in C_i - S$.

In all cases, each of the subpath replacements yields a simple cycle; by applying these replacements for each non-trivial co-component $C_i$, we obtain a simple cycle in $\widehat{G}$ which passes through the vertices $\widehat{x}_1$ and $\widehat{x}_2$, as desired.

($\Longleftarrow$) Suppose that the vertices $x_1, x_2$ belong to the vertex set of the same set $\mathcal{B}_t$ for some $t \in \{1, 2, \ldots, \ell\}$. If $x_1, x_2$ belong to the same non-trivial co-component of $G[M(v)]$, then $x_1, x_2$ belong to the same biconnected component of $\overline{G}$; see Lemma 3.1(iii). Thus, suppose that not both $x_1, x_2$ belong to the same non-trivial co-component of $G[M(v)]$. We will show that $x_1, x_2$ belong to the same biconnected component of $\widehat{G}$. Because of Lemma 3.3 and Step 6 of the algorithm Biconnected_Co-components, the fact that $x_1, x_2 \in V(\mathcal{B}_t)$ implies that the vertices $\widehat{x}_1, \widehat{x}_2$ belong to the same biconnected component of $\widehat{G}$, where for $r = 1, 2$, $\widehat{x}_r = q_i$ if $x_r$ belongs to a non-trivial co-component $C_i$ of $G[M(v)]$, and $\widehat{x}_r = x_r$ otherwise; note that since $x_1, x_2$ do not belong to the same co-component of $G[M(v)]$, $\widehat{x}_1 \neq \widehat{x}_2$. In light of Lemma 3.1(iii), in order to show that $x_1, x_2$ belong to the same biconnected component of $\overline{G}$, it suffices to show that vertices $x'_1, x'_2$ belong to the same biconnected component of $\overline{G}$, where for $r = 1, 2$, $x'_r$ is any vertex in the non-trivial co-component $C_i$ if $x_r \in C_i$ and $x'_r = x_r$ otherwise.

Since $\widehat{x}_1, \widehat{x}_2$ belong to the same biconnected component of $\widehat{G}$, there exists a simple cycle $\widehat{O}$ in $\widehat{G}$ which passes through $\widehat{x}_1, \widehat{x}_2$. As above, we will replace parts of the cycle $\widehat{O}$ by paths in the graph $\overline{G}$, obtaining a simple cycle in $\overline{G}$ which passes through $x'_1, x'_2$, or a collection $O_1, O_2, \ldots, O_r$ of simple cycles such that $x'_1 \in O_1$, $x'_2 \in O_r$, and there exist edges $e_1, e_2, \ldots, e_r$ of $O_1, O_2, \ldots, O_r$ respectively, such that for each $i = 1, 2, \ldots, r-1$, the edges $e_i$ and $e_{i+1}$ belong to the same biconnected component of $\overline{G}$; in either case, we conclude that the vertices $x'_1, x'_2$ belong to the same biconnected component of $\overline{G}$ (in the latter case, Lemma 2.3(iv) implies that the edges of all the cycles $O_1, O_2, \ldots, O_h$ belong to the same biconnected component of $\overline{G}$).

Since the vertex $v$, all the neighbors of $v$ in $G$, and all the vertices inducing trivial co-components of $G[M(v)]$ appear in both $\widehat{G}$ and $\overline{G}$, we need only concentrate in the parts of the cycle $\widehat{O}$ in the subgraphs $\widehat{G}[\widehat{C}_i]$, where $\widehat{C}_i = \{p_i, q_i\} \cup (C_i \cap S)$ are the components of the subgraph of $\widehat{G}$ induced by the neighbors of $v$ and $C_i$ is a non-trivial co-component of $G[M(v)]$ (see Observation 3.1). So, for each component $\widehat{C}_i$ contributing vertices to the cycle $\widehat{O}$, we do the following:

(i) $\widehat{x}_1 = q_i$ *and* $\widehat{x}_2 \notin \widehat{C}_i$*:* Let $\widehat{a}a' \cdots \widehat{x}_2 \cdots b'\widehat{b}$ be the subpath of $\widehat{O}$ passing through $\widehat{x}_2$ such that only $\widehat{a}, \widehat{b}$ belong to $\widehat{C}_i$ (note that the vertex $\widehat{x}_2$ may coincide with the vertices $a'$ or $b'$). For the vertex $\widehat{a}$, we can find a vertex $a \in C_i$ such that $aa' \in E(\overline{G})$: if $\widehat{a} \in S$, then $a = \widehat{a}$, otherwise $\widehat{a} \in \{p_i, q_i\}$ and the definition of the graph $\widehat{G}$ implies that there indeed exists such

a vertex $a$; similarly, for $\widehat{b}$, we can find a vertex $b \in C_i$ such that $bb' \in E(\overline{G})$.
Then, if $a = b$ we replace the part $a'\widehat{a} \cdots \widehat{x}_1 \cdots \widehat{b}b'$ by $a'ab'$, otherwise we replace it by $a'a \cdots bb'$, where the path from $a$ to $b$ is any path in $\overline{G}[C_i]$ connecting $a$ to $b$; in either case, we set $x_1'$ equal to $a$.

(ii) *$\widehat{x}_2 = q_i$ and $\widehat{x}_1 \notin \widehat{C}_i$:* This case is symmetric to case (i).

(iii) *$\widehat{x}_1, \widehat{x}_2 \notin \widehat{C}_i$:* Let $\widehat{a}a' \cdots \widehat{x}_1 \cdots b'\widehat{b}$ and $\widehat{c}c' \cdots \widehat{x}_2 \cdots d'\widehat{d}$ be the subpaths of the cycle $\widehat{O}$ such that none of the vertices of the paths except for $\widehat{a}, \widehat{b}, \widehat{c}, \widehat{d}$ belongs to $\widehat{C}_i$ (again, the two paths can be linked to form a cycle by connecting either $\widehat{a}$ to $\widehat{d}$ and $\widehat{b}$ to $\widehat{c}$, or $\widehat{a}$ to $\widehat{c}$ and $\widehat{b}$ to $\widehat{d}$). As described in case (i), for the vertices $\widehat{a}, \widehat{b}, \widehat{c}, \widehat{d}$ we can find vertices $a, b, c, d \in C_i$, respectively, such that the graph $\overline{G}$ contains the edges $aa'$, $bb'$, $cc'$, and $dd'$. We note that it may be the case that $a = b$ or $c = d$. If $a = b$, then because the cycle $\widehat{O}$ is simple, at least one of $\widehat{a}, \widehat{b}$ is $p_i$ or $q_i$, which implies that at least one of $a', b'$ is non-adjacent in $G$ to at least two vertices in $M(v)$; a similar statement holds for $c', d'$ if $c = d$.
Let us consider the following cases:

- *$a = b$ and $c = d$:* Suppose without loss of generality that $a'$ and $c'$ are non-adjacent in $G$ to at least two vertices in $M(v)$. Then, we break the cycle $\widehat{O}$ into the two cycles $a' \cdots \widehat{x}_1 \cdots b'a$ and $c' \cdots \widehat{x}_2 \cdots d'c$; observe that the edges $aa'$ and $cc'$ belong to the same biconnected component of $\overline{G}$ (Lemma 3.5(ii)).
- *$a = b$ and $c \neq d$:* Suppose without loss of generality that $a'$ is non-adjacent in $G$ to at least two vertices in $M(v)$. Then, we break the cycle $\widehat{O}$ into the two cycles $a' \cdots \widehat{x}_1 \cdots b'a$ and $c' \cdots \widehat{x}_2 \cdots d'd \cdots c$, where the subpath $d \cdots c$ is a path in the subgraph $\overline{G}[C_i]$ from $d$ to $c$; observe that the edges $aa'$ and any edge of the subpath $d \cdots c$ belong to the same biconnected component of $\overline{G}$ (Lemma 3.5(i)).
- *$a \neq b$ and $c \neq d$:* Then, we break the cycle $\widehat{O}$ into the two cycles $a' \cdots \widehat{x}_1 \cdots b'b \ldots a$ and $c' \cdots \widehat{x}_2 \cdots d'd \cdots c$, where the subpaths $b \cdots a$ and $d \cdots c$ are paths in the subgraph $\overline{G}[C_i]$; observe that the edges of the subpaths $b \cdots a$ and $d \cdots c$ belong to the same biconnected component of $\overline{G}$ (Lemma 3.1(iii)).

In all cases, the above subpath replacements yield simple cycles; by applying these replacements for each component $\widehat{C}_i$, we obtain a simple cycle or a collection of simple cycles in $\overline{G}$ with the desired properties. ∎

Let us now compute the time and space complexity of the algorithm. As mentioned, the input graph $G$ is given in adjacency-list representation. Moreover, if $n$ and $m$ are the number of vertices and edges of $G$, recall that the degree of the vertex $v$ is $|N(v)| = O(\sqrt{m})$ (Lemma 2.1) and that the number $k$ of co-components of the subgraph $G[M(v)]$ is $O(\sqrt{m})$ (Lemma 3.1(ii)). Then, the following facts hold:

**F1.** *The cardinality of the set $S$ is $O(\sqrt{m})$.*
Note that the number of elements inserted in $S$ in Steps 1, 3, and 4 is $|N(v)| + 1 = O(\sqrt{m})$, at most $|N(v)| = O(\sqrt{m})$ since at most one vertex is inserted per vertex $u \in N(v)$, and at most $k = O(\sqrt{m})$, respectively.

**F2.** *The number of vertices of the graph $\widehat{G}$ is $O(\sqrt{m})$.*
Follows from Fact F1 and the fact that $k = O(\sqrt{m})$.

**F3.** *The sum of the cardinalities of the edge sets $\widehat{\mathcal{B}}_1, \widehat{\mathcal{B}}_2, \ldots, \widehat{\mathcal{B}}_\ell$ is $O(m)$.*
Follows from the fact that the number of edges of the graph $\widehat{G}$ is $O(m)$, since $V(\widehat{G}) = O(\sqrt{m})$.

**F4.** *The sum of the cardinalities of the edge sets $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_\ell$ is $O(n+m)$.*
Follows from Fact F3 in light of the fact that the vertices $p_i, q_i$, for each co-component $C_i$ of $G[M(v)]$, belong to exactly one set $\widehat{\mathcal{B}}_j$ (Lemma 3.3); then, the vertices in $C_i - S$ belong to exactly one set $\mathcal{B}_j$.

From a data structure point of view, all the graphs are represented by adjacency lists, the sets $N(v)$, $M(v)$, $R$, and $S$ are stored in arrays, and the sets $\widehat{\mathcal{B}}_i$ and $\mathcal{B}_i$, $i = 1, 2, \ldots, \ell$, are stored in linked lists.

Step 1 clearly takes $O(n+m)$ time and $O(n)$ space. Step 2 takes $O(n+m)$ time and space since the co-components of a graph can be computed in time linear in the size of the graph [6, 9, 10, 15]. In Step 3, computing the value of $\mu_u$ for a vertex $u \in N(v)$ takes time proportional to the degree $d(u)$ of $u$ in the graph $G$; moreover, if there is just one non-neighbor of $u$ in $M(v)$, this vertex can be found in $O(d(u)+1)$ time as well. Thus, the execution of Step 3 for all the vertices in $N(v)$ takes $O\big(n + \sum_u (d(u)+1)\big) = O(n+m)$ time. An adjacency-list representation of the subgraph $G[M(v)]$ can be obtained from a copy of the adjacency lists for the graph $G$ in $O(n+m)$ time using $O(n+m)$ space. Since the co-components $C_1, C_2, \ldots C_k$ can be computed in $O(n+m)$ time and space, and the for-loop in Step 4 takes $O(\sqrt{m})$ time, the entire Step 4 is completed in $O(n+m)$ time and space. In Step 5, in order to obtain an adjacency-list representation of the graph $\widehat{G}$, we need to find the pairs of adjacent vertices and link the appropriate records in the adjacency lists. The pairs of adjacent vertices contained in the set $E_1$ can be determined by using a linear-time sorting algorithm to compatibly order the set $S$ and the adjacency lists (for the graph $G$) of the vertices in $S$; then, the vertices of $S$ missing from the adjacency list of each of the vertices in $S$ can be computed in $O(n+m)$ time and space. The pairs of adjacent vertices contained in the sets $E_2$ and $E_3$ can be easily determined and recorded in $O(k) = O(\sqrt{m})$ and $O(n)$ time and space, respectively, while the pairs in $E_4$ can be found by counting, for each vertex $u \in N(v) - R$, its neighbors (in $G$) in each of the sets $C_i$, which can also be done in $O(n+m)$ time and space. Thus, Step 5 can be carried out in $O(n+m)$ time and space. The same holds for Step 6 as well: note that computing the biconnected components of a graph takes time linear in the size of the graph [8], that the vertices $p_i, q_i$ belong to exactly one set $\widehat{\mathcal{B}}_j$ (Lemma 3.3), and that the sum of the cardinalities of the sets $\widehat{\mathcal{B}}_j$ is $O(m)$ (Fact F3). Finally, in light of Fact F4, Step 7 takes $O(n+m)$ time. Summarizing, we have:

**Theorem 3.1.** *Let $G$ be an undirected graph on $n$ vertices and $m$ edges. Then, the algorithm* Biconnected_Co-components *computes the biconnected components of the complement $\overline{G}$ in $O(n+m)$ time and space.*

## 3.1 Parallelizing the Algorithm

In this section, we show how the algorithm Biconnected_Co-components can be efficiently parallelized. As in the description of the sequential co-biconnectivity algorithm, we assume that the input graph is given in adjacency-list representation. We also assume that, for each edge $uv$, the two records in the adjacency lists of $u$ and $v$ are linked together; this helps us re-index the vertices in any subgraph of the given graph fast.

The parallel algorithm executes the exact same steps as the sequential algorithm with the only exception that, for efficiency reasons, we replace the edge set

$$E_4 = \{\, up_i, uq_i \mid \forall \text{non-trivial } C_i, u \in N(v) - R, \text{ and } \exists\, y \in C_i : uy \notin E(G) \,\}$$

in Step 5 of the sequential algorithm by the edge set

$$E'_4 = \{\, up_i, uq_i \mid \forall \text{non-trivial } C_i, u \in N(v) - R, \text{ and } u \text{ and the representative of } C_i \text{ belong to the same co-component of } G[(N(v) - R) \cup M(v)] \,\}.$$

It turns out that this change does not affect the correctness of the algorithm. It is as if we apply the algorithm on a graph $G'$ which is obtained from $G$ after the removal of some edges incident on a vertex $u$ in $N(v) - R$ and a vertex $w$ in $M(v)$ provided that $u, w$ belong to the same co-component of the subgraph $G[(N(v) - R) \cup M(v)]$; equivalently, the complement of $G'$ is obtained from the complement $\overline{G}$ after the addition of these edges. Since the added edges have their endpoints belonging to the same biconnected component of $\overline{G}$, according to Lemma 3.7, this does not alter the biconnected components.

**Lemma 3.7.** *Let $G$ be an undirected graph, $v$ a vertex of $G$, the set $R$ as defined in Step 3 of the algorithm, and let $\widetilde{G}$ be the subgraph $G[(N(v)-R)\cup M(v)]$. Then, if two vertices $a, b \in V(\widetilde{G})$ belong to the same co-component of $\widetilde{G}$, then the vertices $a, b, v$ belong to the same biconnected component of the complement $\overline{G}$.*

*Proof:* It suffices to show that, for any two vertices $x, y$ which are adjacent in the complement of the graph $\widetilde{G}$, the vertices $x, y, v$ belong to the same biconnected component of the complement $\overline{G}$; then, since for any three consecutive vertices $pqr$ in a path in the complement of $\widetilde{G}$, the biconnected components to which $p, q$ and $q, r$ belong share the vertices $q, v$, the lemma follows from Lemma 2.4(iii).

So, let us consider two vertices $x, y$ which are adjacent in the complement of $\widetilde{G}$. If both $x, y$ belong to $M(v)$, then they belong to the same co-component of the subgraph $G[M(v)]$, which has obviously cardinality at least equal to 2, and then by Lemma 3.1(iii) the vertices $x, y, v$ belong to the same biconnected component of $\overline{G}$. If now $x \in N(v)-R$ and $y \in M(v)$, then the construction of the graph $\widetilde{G}$ implies that the vertices $x, y$ are adjacent in $\overline{G}$ as well; then, from the definition of the set $R$, we conclude that, in addition to $y$, there exists at least one other vertex $y' \in M(v)$ such that $xy' \in E(\overline{G})$. But then, the edge set of the subgraph of $\overline{G}$ induced by the vertices $x, y, y', v$ has the biconnectivity property, and thus the vertices $x, y, v$ belong to the same biconnected component of $\overline{G}$. Finally, let us consider the case where $x, y \in N(v)-R$. Again, the definition of the set $R$ implies that there exist vertices $a, b, c, d \in M(v)$ such that $a \neq b$, $c \neq d$, and $\{xa, xb, yc, yd\} \subseteq E(\overline{G})$. But then, no matter whether $\{a, b\} = \{c, d\}$ or whether any of the $c, d$ is different from both $a$ and $b$, the edge set of the subgraph of $\overline{G}$ induced by the vertices $x, y, a, b, c, d, v$ has the biconnectivity property, and thus again the vertices $x, y, v$ belong to the same biconnected component of $\overline{G}$. ∎

Next, we analyze the time and processor complexity of each step of the algorithm on the CREW (or EREW) PRAM model of parallel computation; for details on the PRAM techniques mentioned below, see [1, 16]. We note that augmenting the adjacency-list representation of the input graph so that, for each edge $uv$, it contains pointers linking the record of $u$ in the adjacency list of $v$ and the record of $v$ in the list of $u$, can be easily established in optimal $O(1)$ time using $O(m)$ processors on the EREW PRAM model using an auxiliary array.

*Step 1:* The computation of the degree of a vertex $u$ of the graph $G$ can be done by applying list ranking on the adjacency list of $u$ and by taking the maximum rank; this can be done in $O(\log n)$ time using $O(d(u)/\log n)$ processors on the EREW PRAM, where $d(u)$ denotes the degree of vertex $u$ in $G$. The computation for all the vertices takes $O(\log n)$ time and $O(m/\log n)$ processors on the same model of computation. Locating the vertex $v$ of minimum degree in $G$ can be executed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model. The computation of the vertex sets $N(v)$ and $M(v)$, and the initialization of $S$ can be easily done in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.

*Step 2:* An adjacency-list representation of the graph $G[V(G) - \{v\}]$ can be directly obtained from an adjacency-list representation of $G$ by simply discarding the adjacency list of $v$; note that $v$ has degree 0. The computation of the co-components $C_1, C_2, \ldots, C_k$ of $G[V(G) - \{v\}]$ is done by applying the optimal parallel co-connectivity algorithm of Chong *et al.* [6] which, on a graph on $n$ vertices and $m$ edges, runs in $O(\log n)$ time with $O((n + m)/\log n)$ processors on the EREW PRAM model.

*Step 3:* The set $R$ and the updated set $S$ can be computed as follows: First, for each vertex $u$ in $N(v)$, we compute the number $\mu_u$ of neighbors of $u$ (in $G$) which belong to $M(v)$. If $\mu_u = |M(v)|-1$ or $\mu_u = |M(v)|$ then we insert the vertex $u$ in $R$; if $\mu_u = |M(v)| - 1$, we also insert the vertex $u$ in a set, say, $R'$. Next, for each vertex $w$ in $M(v)$, we compute the number $a_w$ of vertices in $R'$ which are adjacent to $w$; if $a_w < |R'|$ then we insert the vertex $w$ in $S$. It is easy to see that this computation can be carried out within the same time and processor complexity as the computation of the vertex degrees in $G$ (see Step 1); thus, it can be carried out in $O(\log n)$ time using $O(m/\log n)$ processors on the EREW PRAM model.

*Step 4:* An adjacency-list representation of the graph $G[M(v)]$ can be obtained by appropriate processing of copies of the adjacency lists of $G$. Then, re-indexing (based on the ranks of the vertices in the set $M(v)$) is applied to map the vertices in $M(v)$ to consecutive integers starting from 1. To do that, each vertex in $M(v)$ broadcasts its new index number to its adjacency list; next, for each edge, the two adjacency list records associated with it, exchange the new index information. Then, the adjacency-list representation of $G[M(v)]$ can be readily converted into the new indexing scheme. The above computations can be done in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model (see also [6]). The co-components $C_1, C_2, \ldots, C_k$ of the graph $G[M(v)]$ are computed in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model [6].

Then, for each co-component $C_i$, $i = 1, 2, \ldots, k$, checking whether $|C_i| = 1$, and conditionally inserting the only element of $C_i$ in $S$ takes $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model, since $k \leq \sqrt{2m} = O(n)$.

*Step 5:* Having computed the vertex set $S$ and the co-components $C_1, C_2, \ldots, C_k$, the vertex set $V(\widehat{G})$ can be easily computed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model. Let us now see how we compute the edge set $E(\widehat{G})$; it consists of the edge sets $E_1$, $E_2$, $E_3$ and $E'_4$, where the set $E'_4$ is:

$$E'_4 = \{\, up_i, uq_i \mid \forall \text{non-trivial } C_i, u \in N(v) - R, \text{ and } u \text{ and the representative of } C_i \text{ belong to the same co-component of } G[(N(v) - R) \cup M(v)] \,\}.$$

Since $|S| = O(\sqrt{m})$, the set $E_1$ can be computed in $O(\log n)$ time using $O(m/\log n)$ processors on the EREW PRAM model; the computation of the sets $E_2$ and $E_3$ can also be completed within the same time and processor bounds on the same model of computation.

We focus now on the computation of the set $E'_4$. To do that efficiently, we first construct the graph $\widetilde{G} = G[(N(v) - R) \cup M(v)]$. The adjacency-list representation of the graph $\widetilde{G}$ can be obtained in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model (see Step 4). The co-components of the graph $\widetilde{G}$ are computed in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model [6]. Since $|N(v) - R| = O(\sqrt{m})$ and the number $k$ of co-components $C_i$, we can check whether each vertex $u \in N(v) - R$ and the representative of each $C_i$ belong to the same co-component of $\widetilde{G}$ in $O(\log n)$ time using $O(m/\log n)$ processors on the EREW PRAM model.

*Step 6:* The biconnected components of the auxiliary graph $\widehat{G}$ can be computed by applying a parallel algorithm which computes the biconnected components of a graph on $N$ vertices in $O(\log^2 N)$ time using $O(N^2/\log^2 N)$ processors on the CREW PRAM [16]; since the $\widehat{G}$ has $O(\sqrt{m})$ vertices (Fact F2), the biconnected components of $\widehat{G}$ can be computed in $O(\log^2 n)$ time using $O(m/\log^2 n)$ processors on the CREW PRAM model.

Let $\lambda(\widehat{\mathcal{B}})$ be the total number of edges in the biconnected components $\widehat{\mathcal{B}}_1, \widehat{\mathcal{B}}_2, \ldots, \widehat{\mathcal{B}}_\ell$. Since $\lambda(\widehat{\mathcal{B}}) = O(m)$, the edge sets $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_\ell$ can be computed in $O(\log n)$ time using $O(m/\log n)$ processors on the EREW PRAM model. The algorithm which computes the biconnected components $\widehat{\mathcal{B}}_1, \widehat{\mathcal{B}}_2, \ldots, \widehat{\mathcal{B}}_\ell$ generates an output array `b[]` of length $\lambda(\widehat{\mathcal{B}})$ such that `b[`$e$`]` is equal to a representative of the biconnected component containing edge $e$; let $e_1, e_2, \ldots, e_\ell$ be the representatives of the biconnected components.

Let $\gamma(C)$ be the number of vertices in the set $\bigcup_{i=1}^{k} C_i - S$. We construct an array `c[]` of length $\gamma(C)$; the entry `c[`$w$`]`, for each vertex $w \in C_i - S$, corresponds to the edge $vw$ of the complement $\overline{G}$ and is initially assigned the value 0. For each edge $p_i q_i$, $1 \le i \le k$, we compute the representative $e_j$ of the bicomponent $\widehat{\mathcal{B}}_j$ to which $p_i q_i$ belongs; this computation can be done in $O(\log n)$ time using $O(m/\log n)$ processors on the EREW PRAM model (we use prefix sums and array packing techniques [1, 16]; each edge of the graph $\widehat{G}$ has a label $\ell b\ell(\,)$ with the following values: $\ell b\ell(x, y) = \mathit{true}$ if at least one of the vertices $x, y$ is $p_i$ or $q_i$, and $\ell b\ell(x, y) = \mathit{false}$ otherwise). Then, for each vertex $w \in C_i - S$ we assign to `c[`$w$`]` the representative $e_j$ of the biconnected component of $\widehat{G}$ to which the edge $vw$ belongs; this computation can be done in $O(\log n)$ time using $O(m/\log n)$ processors on the EREW PRAM model (to avoid concurrent-read operations we produce $|C_i - S|$ copies of the values $v$ and $e_j$ within the same time and processor bounds on the same model of computation). Finally, in order to remove from $\widehat{\mathcal{B}}_1, \widehat{\mathcal{B}}_2, \ldots, \widehat{\mathcal{B}}_\ell$ the edges incident on $p_i$ or $q_i$, we assign in `b[`$e$`]` the value 0 if $\ell b\ell(x, y) = \mathit{true}$, where $x, y$ are the endpoints of the edge $e$. This computation can be done in $O(1)$ time using $O(m)$ processors on the EREW PRAM model.

We copy the elements of the arrays `b[]` and `c[]` in a new array `d[]` of length $\lambda(\widehat{\mathcal{B}}) + \gamma(C)$, and then delete all the entries of `d[]` with 0 values and move the remaining entries in consecutive locations; the resulting array represents the elements of the edge sets $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_\ell$. This computation can be easily done in $O(\log n)$ time with $O(m/\log n)$ processors on the EREW PRAM model using prefix sums and array packing on the elements of the array `d[]`; see [1, 16].

*Step 7:* Since the sum of the cardinalities of the sets $\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_\ell$ is $O(n+m)$ (Fact F4), returning the array `d[]` can be done in $O(1)$ time using $O(n + m)$ processors.

Taking into consideration the time and processor complexity of each step of the algorithm, and the fact that if a computation can be done in $O(\log n)$ time using $O(p)$ processors, it can also be done in $O(\log^2 n)$ time using $O(p/\log n)$ processors, we obtain the following theorem.

**Theorem 3.2.** *Let $G$ be an undirected graph on $n$ vertices and $m$ edges. Then, the biconnected components of $\overline{G}$ can be computed in $O(\log^2 n)$ time using $O((n + m)/\log^2 n)$ processors on the CREW PRAM model.*

It is interesting to note that with the exception of the computation of the biconnected components of the graph $\widehat{G}$, the rest of the computation can be carried out in $O(\log n)$ time using $O((n + m)/\log n)$ processors on the EREW PRAM model.

# 4 Strongly Connected Components of the Complement of a Graph

In this section, we present a simple optimal sequential algorithm for computing the strongly connected components of the complement $\overline{G}$ of a directed graph $G$, and we show how it can be efficiently parallelized. Our algorithm relies on the following lemma.

**Lemma 4.1.** *Let $G$ be a directed graph, $v$ be a vertex of $G$, and let $A(v) = \{\, x \mid vx \in E(G) \text{ or } xv \in E(G) \,\}$.*

*(i) If $x$ is a vertex of $G$ such that $x \notin A(v)$, i.e., none of the directed edges $vx$ and $xv$ belong to $E(G)$, then the vertices $v$ and $x$ belong to the same strongly connected component of $\overline{G}$.*

*(ii) Let $H_v$ be the directed graph where*

$$\begin{aligned} V(H_v) &= A(v) \cup \{v\}; \\ E(H_v) &= \{\, xy \mid x, y \in A(v) \text{ and } xy \in E(G) \,\} \\ &\quad \cup \{\, vx \mid x \in A(v) \text{ and } \forall\, z \in V(G) - A(v),\ zx \in E(G) \,\} \\ &\quad \cup \{\, xv \mid x \in A(v) \text{ and } \forall\, z \in V(G) - A(v),\ xz \in E(G) \,\}. \end{aligned}$$

*Then, any two vertices $x, y \in V(H_v)$ belong to the same strongly connected component of $\overline{G}$ iff they belong to the same strongly connected component of $\overline{H}_v$.*

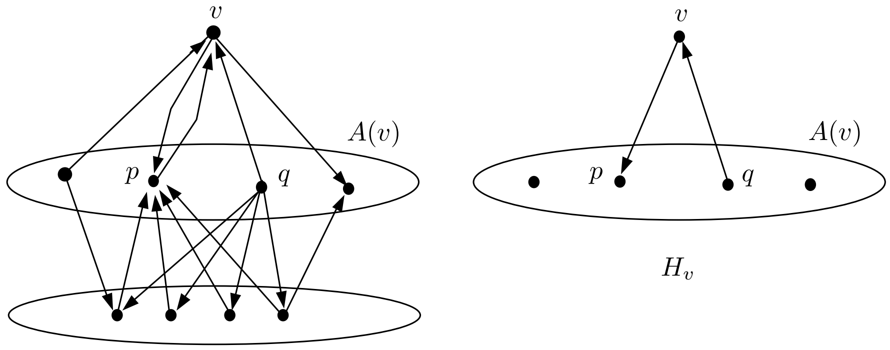


**Figure 1:** A directed graph and the corresponding graph $H_v$.

*Proof:* (i) Trivially true, since the complement $\overline{G}$ contains both the directed edges $vx$ and $xv$.

(ii) ($\Longrightarrow$) (Figure 1 shows a directed graph and the corresponding graph $H_v$.) Suppose that the vertices $x$ and $y$ belong to the same strongly connected component of the complement $\overline{G}$ of $G$; we show that they belong to the same strongly connected component of $\overline{H}_v$. Since the vertices $x, y$ belong to the same strongly connected component of $\overline{G}$, then in $\overline{G}$ there exist a directed path from $x$ to $y$, and a directed path from $y$ to $x$; let $u_0u_1 \cdots u_k$, where $u_0 = x$ and $u_k = y$, and $w_0w_1 \cdots w_\ell$, where $w_0 = y$ and $w_\ell = x$, be the shortest such paths. We distinguish the following two cases:

(a) $x = v$ and $y \in A(v)$: If $u_i \in A(v)\ \forall\, i = 1, 2, \ldots, k$, then the path $vu_1 \cdots u_{k-1}y$ is also a path in $\overline{H}_v$ from $v$ to $y$. Otherwise, since $u_0u_1 \cdots u_k$ is a shortest directed path in $\overline{G}$ from $v$ to $y$, we conclude that $u_1 \in V(G) - (A(v) \cup \{v\})$ and $u_i \in A(v)$ for $2 \leq i \leq k$. Moreover, because $u_1u_2 \in E(\overline{G})$ where $u_1 \notin A(v)$ and $u_2 \in A(v)$, we have that $u_1u_2 \notin E(G)$ and thus $vu_2 \notin E(H_v)$ or equivalently $vu_2 \in E(\overline{H}_v)$. Then, $\overline{H}_v$ contains the directed path $vu_2 \cdots u_{k-1}y$ from $v$ to $y$. In a similar fashion, if $w_i \in A(v)\ \forall\, i = 0, 1, \ldots, \ell - 1$, then the path $yw_1 \cdots w_{\ell-1}v$ is also a path in $\overline{H}_v$ from $y$ to $v$; otherwise, $w_{\ell-1} \in V(G) - (A(v) \cup \{v\})$ and $w_i \in A(v)$ for $0 \leq i \leq \ell - 2$, which implies that $w_{\ell-2}v \in E(\overline{H}_v)$ and thus $\overline{H}_v$ contains the directed path $yw_1 \cdots w_{\ell-2}v$ from $y$ to $v$.

(b) $x, y \in A(v)$: If $u_i \in A(v)\ \forall\, i = 0, 1, \ldots, k$, then the path $xu_1 \cdots u_{k-1}y$ is also a path in $\overline{H}_v$ from $x$ to $y$. Otherwise, let $k' = \min_i\{u_i \notin A(v)\}$; $k'$ is well defined (since there exists a vertex $u_i \notin A(v)$), $k' > 0$, and $u_i \in A(v)$ for $0 \leq i < k'$; moreover, because $u_{k'-1}u_{k'} \in E(\overline{G})$, i.e., $u_{k'-1}u_{k'} \notin E(G)$, and because $u_{k'-1} \in A(v)$ and $u_{k'} \notin A(v)$, we have that $u_{k'-1}v \notin E(H_v)$ or equivalently $u_{k'-1}v \in E(\overline{H}_v)$. Additionally, if $k'' = \max_i\{u_i \notin A(v)\}$, then $k''$ is well defined, $k'' < k$, $u_i \in A(v)$ for $k'' < i \leq k$, and $vu_{k''+1} \in E(\overline{H}_v)$. Then, the vertices $x, u_1, \ldots, u_{k'-1}, v, u_{k''+1}, \ldots, u_{k-1}, y$ form a directed path in $\overline{H}_v$ from $x$ to $y$. In a similar fashion, if $w_i \in A(v)\ \forall\, i = 0, 1, \ldots, \ell$, then the path $yw_1 \cdots w_{\ell-1}x$ is also a path in $\overline{H}_v$ from $y$ to $x$; otherwise, if $\ell' = \min_i\{w_i \notin A(v)\}$ and $\ell'' = \max_i\{w_i \notin A(v)\}$, the vertices $y, w_1, \ldots, w_{\ell'-1}, v, w_{\ell''+1}, \ldots, w_{\ell-1}, x$ form a directed path in $\overline{H}_v$ from $y$ to $x$.

In either case, the vertices $x$ and $y$ belong to the same strongly connected component of $\overline{H}_v$.

($\Longleftarrow$) Suppose that the vertices $x$ and $y$ belong to the same strongly connected component of $\overline{H}_v$; we show that they belong to the same strongly connected component of $\overline{G}$. As above, let $u_0u_1 \cdots u_k$, where $u_0 = x$ and $u_k = y$, and $w_0w_1 \cdots w_\ell$, where $w_0 = y$ and $w_\ell = x$, be the shortest directed paths in $\overline{H}_v$ from $x$ to $y$ and from $y$ to $x$ respectively. Again, we distinguish the following cases:

(a) $x = v$ and $y \in A(v)$: Because the above paths are shortest, then $u_i \in A(v)$ for all $i = 1, 2, \ldots, k$, and $w_j \in A(v)$ for all $j = 0, 1, \ldots, \ell - 1$. Since $vu_1 \in E(\overline{H}_v)$, there exists a vertex $z \in V(G) - A(v)$ such that $zu_1 \in E(\overline{G})$. Similarly, since $w_{\ell-1}v \in E(\overline{H}_v)$, there exists a vertex $z' \in V(G) - A(v)$ such that $w_{\ell-1}z' \in E(\overline{G})$. Then, $\overline{G}$ contains the directed paths $vzu_1 \cdots u_{k-1}y$ (or $vu_1 \cdots u_{k-1}y$ if $z = v$) and $yw_1 \cdots w_{\ell-1}z'v$ (or $yw_1 \cdots w_{\ell-1}v$ if $z' = v$) from $v$ to $y$, and from $y$ to $v$, respectively.

(b) $x, y \in A(v)$: If $u_i \in A(v)$, $\forall\, i = 0, 1, \ldots, k$, then the path $xu_1 \cdots u_{k-1}y$ is also a path in $\overline{G}$. Otherwise, exactly one of the vertices $u_i$ is equal to $v$ (note that if there were more than one such vertices, then clearly there exists a shorter path from $x$ to $y$); let this vertex be $u_j$. Then, $0 < j < k$ and $u_i \neq v$, $\forall\, i \neq j$. Since $u_{j-1}v \in E(\overline{H}_v)$, there exists a vertex $a \in V(G) - A(v)$ such that $u_{j-1}a \in E(\overline{G})$. Similarly, since $vu_{j+1} \in E(\overline{H}_v)$, there exists a vertex $b \in V(G) - A(v)$ such that $bu_{j+1} \in E(\overline{G})$. But then, $\overline{G}$ contains the directed path $xu_1 \cdots u_{j-1}av$ (or $xu_1 \cdots u_{j-1}v$ if $a = v$) from $x$ to $v$, and the directed path $vbu_{j+1} \cdots u_{k-1}y$ (or $vu_{j+1} \cdots u_{k-1}y$ if $b = v$) from $v$ to $y$; thus, there exists a directed path in $\overline{G}$ from $x$ to $y$. Similar statements for the path $yw_1 \cdots w_{\ell-1}x$ imply that either this is a path in $\overline{G}$ (if $w_i \in A(v)$, $\forall\, i = 0, 1, \ldots, \ell$) or there exist an index $j'$ and vertices $a', b' \in V(G) - A(v)$ such that $\overline{G}$ contains the directed paths $yw_1 \cdots w_{j'-1}a'v$ (or $yw_1 \cdots w_{j'-1}v$ if $a' = v$) and $vb'w_{j'+1} \cdots w_{\ell-1}x$ (or $vw_{j'+1} \cdots w_{\ell-1}x$ if $b' = v$); thus, there exists a directed path in $\overline{G}$ from $y$ to $x$ as well.

In either case, the vertices $x, y$ belong to the same strongly connected component of $\overline{G}$. ∎

Our algorithm takes advantage of Lemma 4.1 to reduce the computation of the strongly connected components in the complement $\overline{G}$ to the same computation in the graph $\overline{H}_v$ for a vertex $v$ of $G$ of smallest sum of in-degree and out-degree; the choice of $v$ implies that the graph $\overline{H}_v$ has $O(\sqrt{m})$ vertices (Lemma 2.2), where $m$ is the number of edges of $G$, and thus the strongly connected components in $\overline{H}_v$ can be computed efficiently. Since the strongly connected components of a graph are vertex-disjoint, the algorithm uses an array `sccc[]` of size equal to the number of vertices of the input graph $G$ in which it records the strongly connected components of $\overline{G}$; in particular, sccc[$a$] = sccc[$b$] iff $a, b$ belong to the same strongly connected component of $\overline{G}$. In more detail, the algorithm works as follows:

Algorithm Strong_Co-components
*for the computation of the strongly connected components of the complement of a graph*

*Input:* a directed graph $G$.

*Output:* an array `sccc[]` as described above.

1. $v \leftarrow$ a vertex of $G$ of minimum sum of in-degree and out-degree;
2. if the in-degree and out-degree of $v$ are both equal to 0
   then {*$G$ is a disconnected or a single-vertex graph; $\overline{G}$ is strongly connected*}
   for each vertex $w$ of $G$ do
   sccc[$w$] $\leftarrow v$; {*$v$: representative of the s.c.c of $\overline{G}$*}
   exit;
3. construct the auxiliary graph $H_v$ defined in Lemma 4.1 and, from that, its complement $\overline{H}_v$;
4. compute the strongly connected components of the graph $\overline{H}_v$ and store them in the standard representative-based representation in an array `c[]`;
5. for each vertex $w$ of the graph $G$ do
   if $w \in V(H_v)$
   then sccc[$w$] $\leftarrow$ c[$w$];
   else sccc[$w$] $\leftarrow$ c[$v$];

The correctness of the algorithm follows from Lemma 4.1 and from the fact that if a directed graph has a vertex with in-degree and out-degree both equal to 0, then its complement is strongly connected. Let us now compute the time and space complexity. The input graph $G$ is assumed to be given in adjacency-list representation; let $n$ and $m$ be the number of its vertices and edges. Then, Step 1 can be executed in $O(n+m)$ time, while Step 2 takes $O(n)$ time. An adjacency-list representation of the graph $H_v$ can be obtained from a copy of the adjacency lists of $G$ in $O(n+m)$ time; from that, an adjacency-list representation of its complement $\overline{H}_v$ can be obtained in $O(m)$ time, since the number of vertices of $H_v$ is $O(\sqrt{m})$ (Lemma 2.2). Hence, Step 3 takes $O(n+m)$ time. Step 4 takes $O(m)$ time, since the computation of the strongly connected components of a graph can be done in time linear in the size of the graph [8], and the graph $\overline{H}_v$ has $O(\sqrt{m})$ vertices and thus $O(m)$ edges. Finally, Step 5 takes $O(n)$ time. The space required includes the arrays `sccc[]` and `c[]`, and the adjacency-list representations of the graphs $G$, $H_v$, and $\overline{H}_v$; thus, the algorithm requires $O(n+m)$ space. The following theorem summarizes our results:

**Theorem 4.1.** *Let $G$ be a directed graph on $n$ vertices and $m$ edges. Then, the algorithm* Strong_Co-components *computes the strongly connected components of $\overline{G}$ in $O(n+m)$ time and space.*

## 4.1 Parallelizing the Algorithm

Let us now see how the algorithm Strong_Co-components can be parallelized; we obtain an $O(\log^2 n)$-time and $O(m^{1.188}/\log n)$-processor CREW PRAM solution for the problem. (For details on the PRAM techniques mentioned below (i.e., prefix sums, array packing, list ranking), see [1, 16].

*Step 1:* The computation of the out-degree of a vertex $u$ of the graph $G$ can be done by applying list ranking on the adjacency list of $u$ and by taking the maximum rank; this can be done in $O(\log n)$ time using $O(\text{degree}(u)/\log n)$ processors on the EREW PRAM. The computation for all the vertices takes $O(\log n)$ time and $O(m/\log n)$ processors on the same model of computation. The in-degree of a vertex $u$ can be easily computed by counting the vertices $u$ in all the adjacency lists of the graph $G$; this computation can be done in $O(\log^2 n)$ time with $O(m/\log n)$ processors on the EREW PRAM, using list ranking and sorting on the elements of the adjacency lists of the graph $G$. Locating the vertex $v$ of minimum sum of in-degree and out-degree in $G$ can be executed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM.

*Step 2:* In this step, if the in-degree and out-degree of the vertex $v$ are both equal to 0, the algorithm assigns the value $v$ to each entry of the array `sccc[]`; recall that the size of `sccc[]` equals the number of vertices in the input graph $G$. This assignment can be done in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM (to avoid concurrent-read operations we produce $n$ copies of the value $v$, which can be done within the same time and processor bounds on the same model of computation).

*Step 3:* Let $A(v)$ be the set containing the vertices of $G$ which are adjacent from $v$ and the vertices which are adjacent to $v$; that is, $A(v) = N^+(x) \cup N^-(x)$.

We first construct the subgraph $G_v$ of the graph $G$ induced by the vertex set $A(v) \cup \{v\}$. An adjacency-list representation of the subgraph $G_v$ can be obtained by first processing copies of the adjacency lists of $G$, and re-indexing (based on the ranks of the vertices in the adjacency lists of $G$) so that the vertices in $A(v) \cup \{v\}$ are mapped to consecutive integers starting from 1; then, the adjacency lists of the graph $G_v$ can be obtained in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model (see also [6]).

We next compute the in-degree and the out-degree of the vertices of the graph $G_v$. This computation is done in a fashion similar to that described in Step 1, and it thus can be done in $O(\log^2 n)$ time with $O(m/\log n)$ processors on the EREW PRAM model. The construction of the auxiliary graph $H_v$ defined in Lemma 4.1 can be done using the graph $G_v$ in the following manner ($d^-_{G_v}(x)$ and $d^+_{G_v}(x)$ denote the in-degree and the out-degree of the vertex $x$ in the graph

$G_v$, respectively, whereas the in-degree and the out-degree of the vertex $u$ in the input graph $G$ are denoted by $d^-_G(x)$ and $d^+_G(x)$, respectively):

▷ for each edge $vx$ in $G_v$ do the following:

if $d^-_{G_v}(x) \neq d^-_G(x) - |V(G) - V(G_v)|$ then delete the edge $vx$ from the graph $G_v$;

▷ for each vertex $xv$ in $G_v$ do the following:

if $d^+_{G_v}(x) \neq d^+_G(x) - |V(G) - V(G_v)|$ then delete the edge $xv$ from the graph $G_v$.

It is easy to see that the above construction can be done in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM model.

An adjacency-list representation of the complement $\overline{H}_v$ of the graph $H_v$ can be obtained by first constructing an adjacency matrix for $H_v$, and then by building the appropriate adjacency lists. All this can be carried out in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model, thanks to the fact that $|A(v) \cup \{v\}| = O(\sqrt{m})$ (Lemma 2.2).

*Step 4:* The computation of the strongly connected components of the complement $\overline{H}_v$ of the graph $H_v$ is done by applying a parallel algorithm which computes the strongly connected components of a graph on $N$ vertices in $O(\log^2 N)$ time using $O(N^{2.376}/\log N)$ processors on the CREW PRAM [16]; since the graph $\overline{H}_v$ has $O(\sqrt{m})$ vertices (Lemma 2.2), the execution of the algorithm on $\overline{H}_v$ takes $O(\log^2 n)$ time and requires $O(m^{1.188}/\log n)$ processors on the CREW PRAM model.

It is important to note that the component information needs to be re-indexed back to the original indexing scheme. This can be easily done, while avoiding concurrent reads, by using one copy of the re-indexing array for each vertex in $A(v) \cup \{v\}$; since $|A(v) \cup \{v\}| = O(\sqrt{m})$, the copying can be done in $O(\log m)$ time using $O(m/\log m)$ processors on the EREW PRAM, and the re-indexing in $O(\log n)$ time using $O(\sqrt{m}/\log n)$ processors on the same model of computation.

*Step 5:* Having computed the array `c[]`, that is, the standard representative-based representation of the strongly connected components of the graph $\overline{H}_v$, the computation of the array `sccc[]` can be efficiently done in parallel. Indeed, it is easy to see that the array `sccc[]` can be computed in $O(\log n)$ time using $O(n/\log n)$ processors on the EREW PRAM; again, to avoid concurrent-read operations we produce $|V(G) - V(H_v)| < n$ copies of the value `c[v]` on the same model of computation and within the same time and processor bounds.

Thus, we obtain the following result.

**Theorem 4.2.** *Let $G$ be a directed graph on $n$ vertices and $m$ edges. Then, the strongly connected components of $\overline{G}$ can be computed in $O(\log^2 n)$ time using $O(m^{1.188}/\log n)$ processors on the CREW PRAM model.*

## 4.2 Extending the Approach to Computing the Connected Components

The approach can be extended to the computation of the co-components of an undirected graph in light of the fact that the connected components of such a graph are identical to the strongly connected components of the directed graph that results by replacing each undirected edge by two oppositely directed edges. Then, the following lemma, analogous to Lemma 4.1, holds:

**Lemma 4.2.** *Let $G$ be an undirected graph, $v$ be a vertex of $G$, and let $N(v)$ and $N[v]$ be the neighborhood and closed neighborhood of $v$ in $G$.*

*(i) If $x$ is a vertex of $G$ non-adjacent to $v$, then the vertices $v$ and $x$ belong to the same connected component of $\overline{G}$.*

*(ii) Let $H_v$ be the undirected graph where*

$$V(H_v) = N[v];$$
$$E(H_v) = \{\, xy \mid x, y \in N(v) \text{ and } xy \in E(G) \,\} \cup \{\, vx \mid x \in N(v) \text{ and } \forall\, z \in V(G) - N[v],\ zx \in E(G) \,\}.$$

*Then, two vertices $x, y \in N[v]$ belong to the same connected component of $\overline{G}$ iff they belong to the same connected component of $\overline{H}_v$.*

Then, an algorithm similar to Strong_Co-components yields a very simple optimal sequential algorithm for computing the connected components of the complement of a given graph, in light of Lemma 2.1 and the fact that the connected components of a graph can be computed in time linear in the size of the graph [8].

Finally, a parallelization similar to the one described in Section 4.1 and the algorithm of Chong *et al.* for computing the connected components of a graph on $N$ vertices in $O(\log N)$ time using $O(N^2/\log N)$ processors on the EREW PRAM [4] yields an optimal parallel co-connectivity algorithm simpler than the one in [6]. Therefore, we have:

**Corollary 4.1.** *Let $G$ be an undirected graph on $n$ vertices and $m$ edges. Then, the connected components of $\overline{G}$ can be computed in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model.*

# 5 Concluding Remarks

In this paper we describe sequential co-biconnectivity and strong co-connectivity algorithms which, for a graph on $n$ vertices and $m$ edges, run in $O(n+m)$ time and are therefore optimal. The algorithms are simple, work on the graph (and not on its complement, thus avoiding a potential $\Theta(n^2)$ time complexity), and admit efficient parallelization leading to an optimal $O(\log^2 n)$-time parallel co-biconnectivity algorithm, and an $O(\log^2 n)$-time parallel strong co-connectivity algorithm; both run on the CREW PRAM model of parallel computation. As a byproduct, we obtain a very simple optimal $O(\log n)$-time EREW PRAM co-connectivity algorithm. It is interesting to observe that these results show that, in a parallel process environment, the problems of computing the connected components, the biconnected components, and the strongly connected components can be solved with better time-processor complexity on the complement of a graph rather than on the graph itself.

We leave as an open problem the design of optimal $O(\log n)$-time parallel co-biconnectivity algorithms (note that the connected co-components of a graph can be optimally computed in $O(\log n)$ parallel time on the EREW PRAM model [6]). In particular, finding an $O(\log n)$-time $O(n^2/\log n)$-processor EREW PRAM co-biconnectivity algorithm would be very interesting; note that such an algorithm would imply an optimal $O(\log n)$-time parallel co-biconnectivity algorithm (see Step 6 of our parallel algorithm for the problem).

Our parallel strong co-connectivity algorithm uses a CREW PRAM algorithm for computing the strongly connected components which runs in $O(\log^2 n)$ time with $O(N^{2.376}/\log N)$ processors (see Step 4). It would be worth investigating the existence of cost-optimal or cost-efficient EREW PRAM such algorithms. We also pose this as an open problem.

# References


[1] S.G. Akl, *Parallel Computation: Models and Methods*, Prentice Hall, 1997.

[2] B. Awerbuch and Y. Shiloach, New connectivity and MSF algorithms for ultra-computer and PRAM, *IEEE Trans. Computers* **36**, 1258–1263, 1987.

[3] F.Y. Chin, J. Lam, and I. Chen, Efficient parallel algorithms for some graph problems, *Communications of the ACM* **25**, 659–665, 1982.

[4] K.W. Chong, Y. Han, Y. Igarashi, and T.W. Lam, Improving the efficiency of parallel minimum spanning tree algorithms, *Discrete Applied Math.* **126**, 33–54, 2003.

[5] K.W. Chong, Y. Han, and T.W. Lam, Concurrent threads and optimal parallel minimum spanning trees algorithm, *J. of the ACM* **48**, 297–323, 2001.

[6] K.W. Chong, S.D. Nikolopoulos, and L. Palios, An optimal parallel co-connectivity algorithm, *Theory of Computing Systems* **37**, 527–546, 2004.

[7] R. Cole and U. Vishkin, Faster optimal prefix sums and list ranking, *Information and Computation* **81**, 334–352, 1989.

[8] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms* (2nd edition), MIT Press, Inc., 2001.

[9] E. Dahlhaus, J. Gustedt, and R.M. McConnell, Efficient and practical algorithms for sequential modular decomposition, *J. Algorithms* **41**, 360–387, 2001.

[10] E. Dahlhaus, J. Gustedt, and R.M. McConnell, Partially complemented representation of digraphs, *Discrete Mathematices and Theoretical Computer Science* **5**, 147–168, 2002.

[11] H. Gazit and G.L Miller, An improved parallel algorithm that computes the BFS numbering of a directed graph, *Inform. Process. Letters* **28**, 61–65, 1988.

[12] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[13] D.S. Hirschberg, Parallel algorithms for the transitive closure and the connected components problems, *Proc. 8th ACM Symp. on Theory of Computing (STOC'76)*, 55–57, 1976.

[14] D.S. Hirschberg, A.K. Chandra and D.V. Sarwate, Computing connected components on parallel computers, *Communications of the ACM* **22**, 461–464, 1979.

[15] H. Ito and M. Yokoyama, Linear time algorithms for graph search and connectivity determination on complement graphs, *Inform. Process. Letters* **66**, 209–213, 1998.

[16] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.

[17] M.-Y. Kao, Linear-processor NC algorithms for planar directed graphs I: Strongly connected components, *SIAM J. Computing* **22**, 431–459, 1993.

[18] D. Nath and S.N. Maheshwari, Parallel algorithms for the connected components and minimal spanning trees, *Inform. Process. Letters* **14**, 7–11, 1982.

[19] J. Reif (ed.), *Synthesis of Parallel Algorithms*, Morgan Kaufmann Publishers, San Mateo, California, 1993.

[20] J. Reif, Depth-first search is inherently sequential, *Inform. Process. Letters* **20**, 229–234, 1985.

[21] C. Savage and J. JáJá, Fast, efficient parallel algorithms for some graph problems, *SIAM J. Computing* **10**, 682–691, 1981.

[22] Y. Shiloach and U. Vishkin, An $O(\log n)$ parallel connectivity algorithm, *J. Algorithms* **3**, 57–67, 1982.

[23] R. Tarjan and U. Vishkin, Finding biconnected components and computing tree functions in logarithmic parallel time, *SIAM J. Computing* **14**, 862–874, 1985.

[24] Y.H. Tsin and F.Y. Chin, Efficient parallel algorithms for a class of graph theoretic problems, *SIAM J. Computing* **13**, 580–599, 1984.