Available online at www.sciencedirect.com
ScienceDirect
Discrete Applied Mathematics 155 (2007) 1858–1877
DISCRETE APPLIED MATHEMATICS
www.elsevier.com/locate/dam

On the parallel computation of the biconnected and strongly connected co-components of graphs

Stavros D. Nikolopoulos, Leonidas Palios

Department of Computer Science, University of Ioannina, P.O. Box 1186, GR-45110 Ioannina, Greece

Received 3 October 2004; received in revised form 10 June 2005; accepted 21 July 2006
Available online 27 March 2007

Abstract

In this paper, we consider the problems of co-biconnectivity and strong co-connectivity, i.e., 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. 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. As a byproduct, we obtain a simple optimal $O(\log n)$-time 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 *connected component* of an undirected graph G is a maximal set of vertices of G such that for any two vertices in the set, there exists a path in G connecting them. 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 simple cycle of G [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 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

E-mail addresses: stavros@cs.uoi.gr (S.D. Nikolopoulos), palios@cs.uoi.gr (L. Palios).

0166-218X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2007.03.016

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 [8] and in a parallel process environment [2,3,6,13,14,18,21–23], 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 (DFS)—for both problems run optimally in $O(n+m)$ time [8,12].

By definition, the problems of determining the biconnected and 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 biconnectivity and strong connectivity algorithms 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 DFS 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 DFS 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 DFS. Recently, Chong et al. [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, DFS seems difficult to parallelize; indeed, Reif shows that a restricted version of the approach (lexicographic DFS) is P-complete [19].

Alternatively, there exist several parallel algorithms for the biconnected components problem and the strongly connected components problem that avoid the use of DFS. 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 DFS 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 et al. [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 [1,16,20].

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 et al. [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 $O(n+m)$ sequential 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 simple optimal parallel co-connectivity algo-

rithm. Our proposed algorithms are not data structure-based and they employ neither breadth-first-search nor DFS techniques.

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 discuss possible extensions.

2. Theoretical framework

We consider finite undirected and directed graphs with no loops or multiple 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 (and 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 $\mathscr{E} \subseteq E(G)$ of edges, the subgraph of G *spanned* by $\mathscr{E}$ is denoted $G\langle\mathscr{E}\rangle$.

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 $V(G) - N[x]$ of non-neighbors of x in G. 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, $N^+(x) := \{w \mid xw \in E(G)\}$ is the set of all the vertices of G which are adjacent from x, and $N^-(x) := \{w \mid wx \in E(G)\}$ is the set of all the vertices of G which are adjacent to x; moreover, we use $N(x) := N^-(x) \cup N^+(x)$ and $M(x) := V(G) - (N(x) \cup \{x\})$ to denote the set of all the neighbors and all the non-neighbors of x in G, respectively. 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 $v_0 v_1 \cdots v_k$ of a graph G is a *path* from v_0 to v_k (or a v_0–v_k path) of *length* k in G provided that $v_{i-1} v_i \in E(G)$ for $i = 1, 2, \ldots, k$. A *cycle* (or *closed path*) of length $k+1$ is a path $v_0 v_1 \cdots v_k$ such that $v_k v_0 \in E(G)$. A path (cycle) is undirected or directed depending on whether G is undirected or directed. A path (cycle) is called *simple* if none of its vertices occurs more than once; it is called *trivial* if $k = 0$. An undirected graph G is *connected* if between any two vertices u and v there exists a u–v path in G. A directed graph G is *strongly connected* if for any two vertices u and v of G, there exists both a u–v path and a v–u path in G.

A *connected component* (or component) of an undirected graph G is a maximal set of vertices, say, $C \subseteq V(G)$, such that for every pair of vertices $x, y \in C$, there exists a x–y path in the subgraph $G[C]$ of G induced by the vertices in C. 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 simple cycle of G [8]. In particular, we give the following definition:

Definition 2.1. A set $\mathscr{E}$ of edges of an undirected graph G *has the biconnectivity property* if every pair of edges in $\mathscr{E}$ lie on a common simple cycle in the subgraph of G spanned by the set $\mathscr{E}$.

Clearly, a biconnected component of G is a maximal set of edges of G that have the biconnectivity property; moreover, all the edges of G which belong to a set having the biconnectivity property belong to the same biconnected component of G. A *cutpoint* (or *articulation point*) of G is a vertex whose removal increases the number of connected components of G (i.e., the removal of the vertex disconnects a component of G), and a *bridge* is an edge with this property. By definition, a biconnected component contains no cutpoints or bridges. The *biconnected co-components* (or co-bicomponents) of a graph 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 such that for any two vertices x and y in the set, there exists both a (directed) x–y path and a (directed) y–x path in the subgraph of G induced by the vertices in the set; the *strongly connected co-components* (or strong 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 (*Chong et al. [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 is less than $2\sqrt{m}$.*

Proof. Since v is G's vertex of minimum sum of in-degree and out-degree, then for the sum of in-degrees and out-degrees over all vertices x of G, which is equal to $2m$, we have that

$$2m = \sum_x (d^-(x) + d^+(x)) \geqslant n \cdot (d^-(v) + d^+(v)) \Longrightarrow d^-(v) + d^+(v) \leqslant 2m/n;$$

then, if we take into account that $n > \sqrt{m}$ because $m \leqslant n \cdot (n-1) < n^2$, we get that $d^-(v) + d^+(v) < 2\sqrt{m}$. □

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

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

(i) *for every edge $e \in \mathscr{E}$ and any two vertices $u, w \in V_{\mathscr{E}}$, the subgraph $G\langle\mathscr{E}\rangle$ of G spanned by the edges in $\mathscr{E}$ contains a simple path from u to w that passes through e;*
(ii) *for any simple path in G with endpoints in $V_{\mathscr{E}}$ and edge set $\mathscr{P}$, the set $\mathscr{E} \cup \mathscr{P}$ has the biconnectivity property;*
(iii) *for any set $\mathscr{E}' \subseteq E(G)$ such that $\mathscr{E}'$ has the biconnectivity property and $\mathscr{E} \cap \mathscr{E}' \neq \emptyset$, the set $\mathscr{E} \cup \mathscr{E}'$ also has the biconnectivity property.*

Proof. (i) Since the set $\mathscr{E}$ has the biconnectivity property, there exists a simple path, say, ρ, in the graph $G\langle\mathscr{E}\rangle$ from u to w. Additionally, if e' is any edge of ρ, there exists a simple cycle in $G\langle\mathscr{E}\rangle$ 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 ρ and ρ' as we move from a to b along ρ'; because the two paths share the edge e', the vertices x, y are well defined and distinct (note that x or y may coincide with a, b). Then, if $\rho = u \cdots x \cdots y \cdots w$, the path that consists of the part of ρ from u to x, followed by the part of ρ' from x to a, followed by the edge e, followed by the part of ρ' from b to y, followed by the part of ρ 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 ρ from u to y, followed by the part of ρ' from y to b, followed by the edge e, followed by the part of ρ' from a to x, followed by the part of ρ from x to w.

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

Suppose now that there exist vertices of the path ρ, other than its endpoints, that belong to $V_{\mathscr{E}}$. Let us remove from ρ any edges in $\mathscr{E}$ and let us break the resulting subpaths at their vertices that belong to $V_{\mathscr{E}}$. We obtain a collection of simple paths $\rho_1, \rho_2, \ldots, \rho_k$. Then, as shown earlier, the union of $\mathscr{E}$ and the edge set of ρ_1 has the biconnectivity property; in turn, the union of this set with the edge set of ρ_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 $\mathscr{E} \cup \mathscr{P}$ has the biconnectivity property.

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

Lemma 2.3 has the following interesting implications.

Lemma 2.4. *Let G be an undirected graph, let $\mathscr{E}_1, \mathscr{E}_2 \subseteq E(G)$ 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 $\mathscr{E}_1$ and $\mathscr{E}_2$, respectively.*

(i) *The edge set of the subgraph of G induced by V_1 (or V_2) has the biconnectivity property.*
(ii) *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.*
(iii) *Suppose that $V_1 \cap V_2 = \{v\}$. If there exists a simple path ρ 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.*
(iv) *If $|V_1 \cap V_2| \geqslant 2$, then the edge set of the subgraph of G induced by $V_1 \cup V_2$ has the biconnectivity property.*

Proof. (i) Trivially true due to Lemma 2.3(ii), since any edge $e \notin \mathscr{E}_1$ whose endpoints belong to V_1 constitutes a simple path connecting two vertices in V_1.

(ii) In light of Lemma 2.4(i), it suffices to show that the set $\mathscr{E} = \mathscr{E}_1 \cup \mathscr{E}_2 \cup \{ux, wy\}$ has the biconnectivity property. The biconnectivity of $\mathscr{E}_1$ and $\mathscr{E}_2$ implies that for any two edges in $\mathscr{E}_1$ or $\mathscr{E}_2$, there exists a simple cycle in the subgraph $G\langle\mathscr{E}\rangle$ spanned by the edges in $\mathscr{E}$ which passes through them. Additionally, for any edge $e_1 \in \mathscr{E}_1$ and any edge $e_2 \in \mathscr{E}_2$, Lemma 2.3(i) implies that there exist paths of edges in $\mathscr{E}_1$ and $\mathscr{E}_2$ 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 $G\langle\mathscr{E}\rangle$ 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 $e \in \mathscr{E}_1 \cup \mathscr{E}_2$.

(iii) It suffices to show that the set $\mathscr{E}_1 \cup \mathscr{E}_2 \cup \mathscr{P}$ has the biconnectivity property, where $\mathscr{P}$ is the edge set of the path ρ. Let $u \in V_1$ and $x \in V_2$ be the endpoints of ρ, and let y be the first vertex in V_2 that we meet as we move along ρ from u to x; we break the path ρ at y and we obtain two simple paths ρ_1 (with endpoints u and y) and ρ_2 (with endpoints y and x). The biconnectivity of $\mathscr{E}_2$ implies that there exists a simple path from y to v along edges in $\mathscr{E}_2$. By concatenating ρ_1 and this path we obtain a simple path ρ' whose both endpoints belong to V_1; then, Lemma 2.3(ii) implies that the union of the edge set $E(\rho')$ of ρ' and $\mathscr{E}_1$ has the biconnectivity property. In turn, this implies that the set $\mathscr{E}_1 \cup \mathscr{E}_2 \cup E(\rho') = \mathscr{E}_1 \cup \mathscr{E}_2 \cup E(\rho_1)$ has the biconnectivity property since the path ρ does not go through v, i.e., $y \neq v$, and thus the sets $\mathscr{E}_2$ and $E(\rho')$ share edges (Lemma 2.3(iii)). Finally, since the endpoints of the path ρ_2 belong to the vertex set of $\mathscr{E}_1 \cup \mathscr{E}_2 \cup E(\rho_1)$, its edge set can also be included and the resulting set has the biconnectivity property (Lemma 2.3(ii)).

(iv) It suffices to show that the set $\mathscr{E}_1 \cup \mathscr{E}_2$ has the biconnectivity property. Let $x, y \in V_1 \cap V_2$, where $x \neq y$. Since the set $\mathscr{E}_1$ has the biconnectivity property, there exists a simple path ρ of edges in $\mathscr{E}_1$ from x to y (Lemma 2.3(i)). Then, Lemma 2.3(ii) implies that the union $\mathscr{A} = \mathscr{E}_2 \cup \mathscr{P}$, where $\mathscr{P}$ is the edge set of the path ρ, has the biconnectivity property; finally, Lemma 2.3(iii) completes the proof since the sets $\mathscr{A}$ and $\mathscr{E}_1$ each have the biconnectivity property and they are not disjoint. □

3. Biconnected co-components

Let G be an undirected graph on n vertices and m edges. In this section, we 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 results established in the following two lemmata.

Lemma 3.1. *Let G be an undirected graph on $m \geqslant 1$ edges and x be a vertex of G. If $C_1, C_2, \ldots, C_k$ are the co-components of the subgraph $G[M(x)]$ of 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 non-trivial co-component C_i (i.e., $|C_i| \geqslant 2$), the edges of the subgraph of the complement $\overline{G}$ induced by*

$$\{x\} \cup C_i \cup \{u \in N(x) \mid |M(x) - N(u)| \geqslant 2 \text{ and } |C_i - N(u)| \geqslant 1\}$$

(*i.e., the vertices in* $\{x\} \cup C_i$ *and those among the vertices in* $N(x)$ *that are not adjacent to at least* 1 *vertex in* C_i *and to at least one additional vertex in* $M(x)$*) belong to the same biconnected component of* $\overline{G}$.

Proof. (i) Follows from the definition of connected components.

(ii) If $k=1$, the lemma clearly holds, since $m \geqslant 1$. Suppose now that $k \geqslant 2$. Since $C_1, C_2, \ldots, C_k$ are the co-components of $G[M(x)]$, then every vertex in C_i is adjacent in G to every vertex in C_j, for all $j \neq i$. This implies that G contains at least $k(k-1)/2$ edges, and thus $k(k-1)/2 \leqslant m$. Since $k^2/4 \leqslant k(k-1)/2$ for $k \geqslant 2$, we have that $k^2/4 \leqslant m$. The lemma follows.

(iii) First, we show that for each non-trivial co-component C_i, the edge set of the subgraph $\overline{G}[C_i \cup \{x\}]$ has the biconnectivity property. For any edge $e = yz$ of $\overline{G}[C_i]$, the cycle xyz in $\overline{G}$ implies that 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 DFS 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(iv), since the vertex sets of the two merged edge sets share at least two elements, namely, x and at least one of y', z'. Then, the fact that the edge set of $\overline{G}[C_i \cup \{x\}]$ has the biconnectivity property follows from the fact that the graph $\overline{G}[C_i]$ is connected.

Next, it suffices to prove that, for any vertex $u \in N(x)$ such that u is non-adjacent in G to vertices $w \in C_i$ and $w' \in M(x) - \{w\}$, the edges of the subgraph $\overline{G}[\{x, u\} \cup C_i]$ belong to the same biconnected component. The pairs uw, uw', xw, xw' are edges in $\overline{G}$ which form the cycle $uwxw'$, and thus the edge set $\{uw, uw', xw, xw'\}$ of $\overline{G}$ has the biconnectivity property; as this edge set and the edge set of $\overline{G}[\{x\} \cup C_i]$ share the edge xw, Lemma 2.3(iii) implies that the edges of $\overline{G}[\{x, u\} \cup C_i]$ indeed belong to the same biconnected component. □

Lemma 3.1 implies that the non-neighbors of a vertex x in G (on n vertices and m edges), which may be as many as $\Theta(n)$, participate in $O(\sqrt{m})$ biconnected components in the complement $\overline{G}$. Thus, it might be possible to retain only $\Theta(\sqrt{m})$ vertices among the non-neighbors of x while still not loosing information on the biconnected components of $\overline{G}$. Additionally, if we chose x to be a minimum degree vertex v in G, which according to Lemma 2.1 has $O(\sqrt{m})$ neighbors, then the total number of vertices to retain would be $O(\sqrt{m})$. Of course, the cutpoints and the bridge endpoints in $\overline{G}$ need to be retained. Although finding these points seems to necessitate the computation of the biconnected components of $\overline{G}$, a set S containing them can be easily constructed. In particular, we define:

$$S = \{v\} \cup N(v) \tag{1}$$

$$\cup \{y \mid y \text{ forms a trivial co-component of } G[M(v)]\} \tag{2}$$

$$\cup \{w \mid w \in M(v) \text{ s.t. } \exists u \in N(v) \text{ for which } M(v) - N(u) = \{w\}\}, \tag{3}$$

i.e., in addition to the entire closed neighborhood of v and the vertices forming trivial co-components of $G[M(v)]$, the set S contains each vertex $w \in M(v)$ which is the only non-neighbor of some neighbor of v. Lemma 3.2 establishes the fact that this set S indeed contains the cutpoints and the bridge endpoints in $\overline{G}$.

Lemma 3.2. *Let* G *be an undirected graph and let* S *be the set of vertices described above. 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 the set S contains the entire closed neighborhood $N[v]$ of v in G (see Eq. (1)), we need consider only the case that x belongs to the set $M(v)$ 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 (i.e., trivial) co-component of $G[M(v)]$. But then, x is contained in the set S; see Eq. (2).

Suppose now that the vertex x belongs to a connected component A of $\overline{G}$ and that x's removal disconnects the subgraph $\overline{G}[A]$. Then, $M(v) \cup \{v\} \subseteq A$, as all the vertices in $M(v)$ are connected in $\overline{G}$ through v. In fact, A contains vertices from $N(v)$ as well, for otherwise the removal of x would not disconnect $\overline{G}[A]$. When x is removed, the vertices in $(M(v) - \{x\}) \cup \{v\}$ (and perhaps some vertices in $N(v)$ as well) remain connected, yet there exists a subset A' of vertices in A which are no longer connected in $\overline{G}$ to the vertices in $(M(v) - \{x\}) \cup \{v\}$. Hence, $A' \subseteq N(v)$ and every

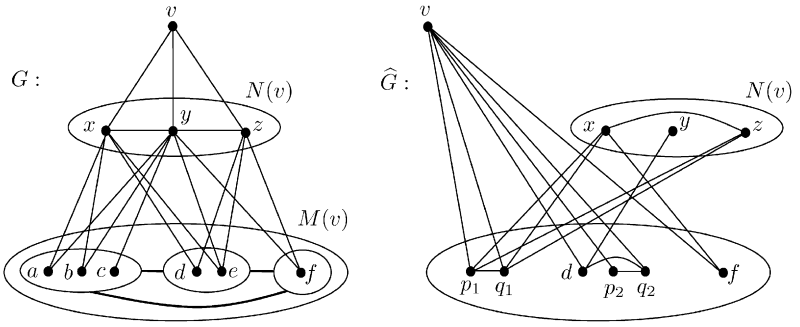

Fig. 1. A graph G and the corresponding graph $\widehat{G}$.

vertex in $M(v) - \{x\}$ is adjacent in G to every vertex in A'. But then, if u is a vertex in A', 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 contained in the set S; see Eq. (3). □

Then, for each non-trivial co-component C_i of $G[M(v)]$ (i.e., $|C_i| \geqslant 2$), we replace the vertices not in the set S by two new vertices p_i, q_i with appropriate adjacencies to vertices in S so that the resulting graph maintains all the information on the biconnected components of $\overline{G}$ and only that. More specifically, if we define

$$R = \{u \mid u \in N(v) \text{ and } |M(v) - N(u)| \leqslant 1\} \tag{4}$$

(i.e., the set R is the set of all neighbors of v which are adjacent to either all the non-neighbors of v or all but one non-neighbor of v in G), the algorithm uses an auxiliary graph $\widehat{G}$, defined as follows:

$$V(\widehat{G}) = S \cup \{p_i, q_i \mid C_i \text{ is non-trivial}\},$$
$$E(\widehat{G}) = E_1 \cup E_2 \cup E_3 \cup E_4,$$

where

$$E_1 = \{xy \mid (x, y \in N(v) \text{ or } x \in N[v], y \in M(v) \cap S) \text{ and } xy \notin E(G)\},$$
$$E_2 = \{vp_i, vq_i, p_iq_i \mid C_i \text{ is non-trivial}\},$$
$$E_3 = \{wq_i \mid C_i \text{ is non-trivial and } w \in C_i \cap S\},$$
$$E_4 = \{up_i, uq_i \mid C_i \text{ is non-trivial and } u \in N(v) - R \text{ such that } \exists\, y \in C_i : uy \notin E(G)\}.$$

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 have been replaced by the vertices p_i, q_i: the edge set E_1 contains the edges of $\overline{G}$ connecting pairs of vertices in S except if both vertices belong to $M(v)$, the sets E_2 and E_4 are motivated by Lemma 3.1(iii), and the set E_3 along with the edge p_iq_i of E_2 ensures the connectivity of each of the sets $\{p_i, q_i\} \cup (C_i \cap S)$. Moreover, note that if in G a neighbor u of v has exactly one non-neighbor, say, w, belonging to $M(v)$, then S contains both u and w so that $u, w \in V(\widehat{G})$ and $uw \in E(\widehat{G})$. Fig. 1 shows a graph G and the corresponding graph $\widehat{G}$: Note that vertex f of G forms a trivial co-component of $G[M(v)]$, and vertices x, y, and z are non-adjacent to two, one, and three vertices in $M(v)$, respectively;thus, in this case, $R = \{y\}$ and $S = N[v] \cup \{d, f\}$. Then, the edge setsE_1, E_2, E_3, and E_4 of the graph $\widehat{G}$ shown in the figure are: $E_1 = \{vd, vf, xz, xf, yd\}$, $E_2 = \{vp_1, vq_1, p_1q_1, vp_2, vq_2, p_2q_2\}$, $E_3 = \{dq_2\}$, and $E_4 = \{xp_1, xq_1, zp_1, zq_1\}$. The graph $\widehat{G}$ has two biconnected components, namely, the edge sets of the subgraphs induced by $\{v, x, z, p_1, q_1, f\}$ and $\{v, d, p_2, q_2\}$, whereas the edge yd is a bridge; one can verify that the graph G has two biconnected co-components, the edge sets of the subgraphs of $\overline{G}$ induced by $\{v, x, z, a, b, c, f\}$ and $\{v, d, e\}$, and the edge yd is again a bridge of $\overline{G}$.

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 $\mathscr{B}_i$ of its edges such that every vertex of the component is incident on at least one edge in $\mathscr{B}_i$. 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 $\mathscr{B}_i$. (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 $\mathscr{B}_i$ 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: sets of edges of the complement $\overline{G}$ such that all edges of the subgraph of $\overline{G}$ spanned by each such set form a biconnected component of $\overline{G}$.

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 $C_1, C_2, \ldots, C_t$ 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., $|C_i| \geqslant 2$);
   `exit`;
3. $R \leftarrow \emptyset$;
   `for` each vertex u in $N(v)$ `do`
   compute the number μ_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; {*potential cutpoint in $\overline{G}$*}
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 in $\overline{G}$*}
5. construct the auxiliary graph $\widehat{G}$ defined earlier in this section in terms of the graph G, the co-components $C_1, C_2, \ldots, C_k$, and the computed sets R and S;
6. compute the biconnected components (edge sets) $\widehat{\mathscr{B}}_1, \widehat{\mathscr{B}}_2, \ldots, \widehat{\mathscr{B}}_\ell$ of $\widehat{G}$;
   `for` each $i = 1, 2, \ldots, \ell$ `do`
   $\mathscr{B}_i \leftarrow \widehat{\mathscr{B}}_i$;
   `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 component $\widehat{\mathscr{B}}_j$
   `then for` each vertex $w \in C_i - S$ `do`
   insert the edge vw in $\mathscr{B}_j$;
   remove from $\mathscr{B}_j$ all edges incident on p_i or q_i;
7. print the resulting edge sets $\mathscr{B}_1, \mathscr{B}_2, \ldots, \mathscr{B}_\ell$; 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 $\mathscr{B}_1, \mathscr{B}_2, \ldots, \mathscr{B}_\ell$;

We note that the set S collected by the algorithm contains precisely the required vertices: the vertices in Eq. (1) are collected in Step 1, those in Eq. (2) in Step 4, and those in Eq. (3) in Step 3. Additionally, the set R indeed contains the vertices in $N(v)$ which have at most one non-neighbor in G belonging to $M(v)$ in accordance with Eq. (4). Before proving the correctness of the algorithm, we state and prove an important fact about the auxiliary graph $\widehat{G}$.

Lemma 3.3. *Let G be an undirected graph, v a vertex of G, and the graph $\widehat{G}$ as defined above. Then, for each non-trivial co-component C_i of the graph $G[M(v)]$, the vertices p_i, q_i belong to the vertex set of 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 $\{v, p_i, q_i\} \cup (C_i \cap S)$ 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 $w \in C_i \cap S$ (Lemma 2.3(iii)). Thus, there exists a biconnected component of $\widehat{G}$ whose vertex set contains $\{v, p_i, q_i\} \cup (C_i \cap S)$. The proof will be complete if we show that neither p_i nor q_i belong to the vertex sets of more than one biconnected component of $\widehat{G}$; to do that, we show that any component containing p_i contains q_i as well and that any component containing q_i contains p_i as well, and then, according to Lemma 2.4(iv), such a component is uniquely identified.

Let $\widehat{\mathscr{B}}$ be any biconnected component of $\widehat{G}$ containing p_i; then, there exists an edge, say, $p_i w$, in $\widehat{\mathscr{B}}$. If $w = q_i$, we are done; otherwise, because in $\widehat{G}$ the vertex q_i is adjacent to all the neighbors of p_i (except for q_i), $\widehat{G}$ contains the simple cycle $p_i w q_i$, which implies that q_i belongs to the vertex set of $\widehat{\mathscr{B}}$. Next, let $\widehat{\mathscr{B}}'$ be any biconnected component of $\widehat{G}$ containing q_i; then, there exists an edge, say, $q_i w$, in $\widehat{\mathscr{B}}'$. If $w = p_i$, we are again done; otherwise, if $w \in C_i \cap S \cap \{v\}$ (i.e., $q_i w \in E_2 \cup E_3$), then $\widehat{\mathscr{B}}'$ contains p_i as well, because its vertex set shares two vertices with the vertices in $\{v, p_i, q_i\} \cup (C_i \cap S)$ (Lemma 2.4(iv)), whereas if $w \in N(v) - R$ (i.e., $q_i w \in E_4$), then p_i is also adjacent to w and $\widehat{G}$ contains the simple cycle $p_i w q_i$, again implying that p_i belongs to the vertex set of $\widehat{\mathscr{B}}'$. □

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 $\widehat{G}$ that are not incident on any p_i or q_i are edges of the complement $\overline{G}$, from Lemma 3.3, and from Lemma 3.5 with the aid of Lemma 3.1.

Lemma 3.4. *Let G be an undirected graph, v be a vertex of G whose degree is 0, and let $C_1, C_2, \ldots, C_t$ be the co-components of the subgraph $G[V(G) - \{v\}]$. 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 C_i (i.e., $|C_i| \geqslant 2$), $1 \leqslant i \leqslant t$.*

Proof. Clearly, the vertices of the graph G other than v are partitioned in the sets $C_1, C_2, \ldots, C_t$. 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 $|C_j| \geqslant 2$, 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. *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. The proof proceeds by showing that if two edges $e_1, e_2 \in E(\overline{G})$ belong to the same biconnected component in $\overline{G}$, then the "corresponding" edges $\widehat{e}_1, \widehat{e}_2 \in E(\widehat{G})$ also belong to the same biconnected component in $\widehat{G}$, and vice versa. The correspondence between edges of $\overline{G}$ and edges of $\widehat{G}$ is defined as follows:

C1. For an edge $e = uw \in E(\overline{G})$, the edge $\widehat{e} \in E(\widehat{G})$ corresponding to e is:

(a) if $u, w \in N(v)$, or $u \in N[v]$ and w forms a trivial co-component of $G[M(v)]$, or $u \in R$ and $w \in C_i \cap S$ for some non-trivial co-component C_i of $G[M(v)]$, then $\widehat{e} = e$;
(b) if $u \in N[v] - R$ and $w \in C_i$ for some non-trivial co-component C_i of $G[M(v)]$, then $\widehat{e} = up_i$ or $\widehat{e} = uq_i$;
(c) if e is any edge of $\overline{G}[C_i]$ for some non-trivial co-component C_i of $G[M(v)]$, then $\widehat{e}$ is any edge of $\widehat{G}[\{p_i, q_i\} \cup (C_i \cap S)]$.

C2. For an edge $\widehat{e} = uw \in E(\widehat{G})$, the edge $e \in E(\overline{G})$ corresponding to $\widehat{e}$ is:

(a) if $u, w \in N(v)$, or $u \in N[v]$ and w forms a trivial co-component of $G[M(v)]$, or $u \in R$ and $w \in C_i \cap S$ for some non-trivial co-component C_i of $G[M(v)]$, then $e = \widehat{e}$;

(b) if $u \in N[v] - R$ and $w \in \{p_i, q_i\} \cup (C_i \cap S)$ for some non-trivial co-component C_i of $G[M(v)]$, then $e = uy$ where $y \in C_i$;
(c) if $\widehat{e}$ is any edge of $\widehat{G}[\{p_i, q_i\} \cup (C_i \cap S)]$ for some non-trivial co-component C_i of $G[M(v)]$, then e is any edge of $\overline{G}[\{v\} \cup C_i]$.

First, note that all the edges of $\overline{G}$ have been considered in the correspondence scheme C1 and all the edges of $\widehat{G}$ have been considered in the correspondence scheme C2. Additionally, note that for a non-trivial co-component C_i of $G[M(v)]$, all the edges of $\overline{G}[\{v\} \cup C_i]$ belong to the same biconnected component of $\overline{G}$ (Lemma 3.1(iii)), that the edges up_i and uq_i, where $u \in N[v] - R$, belong to the same biconnected component of $\widehat{G}$ (because up_iq_i is a simple cycle in $\widehat{G}$; see edge set E_4), and that all the edges of $\widehat{G}[\{p_i, q_i\} \cup (C_i \cap S)]$ also belong to the same biconnected component of $\widehat{G}$ (Lemmata 3.3 and 2.4(i)).

($\Longrightarrow$) Let $e_1, e_2 \in E(\overline{G})$ be two edges belonging to the same biconnected component of $\overline{G}$; we will show that the corresponding edges $\widehat{e}_1, \widehat{e}_2 \in E(\widehat{G})$ belong to the same biconnected component of $\widehat{G}$. Since e_1, e_2 belong to the same biconnected component of $\overline{G}$, there exists a simple cycle, say, O, in $\overline{G}$ that passes through them. 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 $\widehat{e}_1$ and $\widehat{e}_2$. Since all the vertices in $N[v]$ as well as those in $M(v)$ forming trivial co-components of the subgraph $G[M(v)]$ appear in $\widehat{G}$, we need only concentrate in the parts of the cycle O in the subgraphs of $\overline{G}$ induced by the non-trivial co-components of$G[M(v)]$. So, we process in turn each such co-component contributing vertices to the cycle O.

Let C_i be a non-trivial co-component of the subgraph $G[M(v)]$. It is important to observe that the following property holds:

P1. Let aa' be an edge of the complement $\overline{G}$ such that $a \in C_i$ and $a' \notin C_i$, for a non-trivial co-component C_i of $G[M(v)]$; then, clearly $a' \in N[v]$. Moreover,

- if $a' \in R$, then $a \in C_i \cap S$ and $a'a \in E(\widehat{G})$;
- if $a' \in N[v] - R$, then $a'p_i, a'q_i \in E(\widehat{G})$.

Note that if $a' \in R$, then a' would have at most one non-neighbor belonging to $M(v)$ in G; because $aa' \in E(\overline{G})$, a' would have exactly one non-neighbor in $M(v)$, which would precisely be a, and thus $a \in S$. The fact that $a'a \in E(\widehat{G})$ follows from the definition of the edge set E_1. In turn, if $a' \in N[v] - R$, then the fact that $a'p_i, a'q_i \in E(\widehat{G})$ follows from the definitions of the edge sets E_2 (if $a' = v$) and E_4 (if $a' \in N(v) - R$).

Since all the edges of the subgraph $\widehat{G}[\{p_i, q_i\} \cup (C_i \cap S)]$ belong to the same biconnected component (Lemma 3.3) and the algorithm correctly puts all the edges of the subgraph $\overline{G}[\{v\} \cup C_i]$ in the same biconnected component (Lemma 3.3 and Step 6), it suffices to consider edges e_1, e_2 of $\overline{G}$ that do not have all their four endpoints in C_i. Thus, we distinguish the following cases:

(i) *Exactly one of the edges e_1, e_2 has both its endpoints in the co-component C_i*: Suppose that this edge is e_1. Then, if $e_2 = x_2y_2$, let $aa' \cdots x_2y_2 \cdots b'b$ be the subpath of O passing through e_2 such that only a, b belong to C_i (note that the edge e_2 may coincide with the edges aa' or $b'b$). Clearly, $a', b' \in N[v]$ (Property P1), yet not both a' and b' belong to R, for otherwise $a = b$, in contradiction to the fact that the cycle O is simple and passes through the edge e_1 which has both its endpoints in C_i. Thus, assume without loss of generality that $a' \in N[v] - R$; then, a' is adjacent to p_i and q_i (Property P1). Then, we replace the part $a'a \cdots x_1y_1 \cdots bb'$ of the cycle O, which passes through $e_1 = x_1y_1$, by the path $a'q_ibb'$ if $a' \in N[v] - R$ and $b' \in R$, or the path $a'p_iq_ib'$ if $a', b' \in N[v] - R$. In each case, the edge e_1 is replaced by an edge bq_i or p_iq_i of $\widehat{G}[\{p_i, q_i\} \cup (C_i \cap S)]$, while the edge aa' is replaced by $a'q_i$ or $a'p_i$ and the edge bb' remains unchanged if $b' \in R$ or else is replaced by $b'q_i$; in all cases, our correspondence scheme C1 is followed.

(ii) *None of the edges e_1, e_2 has both its endpoints in the co-component C_i*: If the edges $e_1 = x_1y_1$ and $e_2 = x_2y_2$ belong to a path $aa' \cdots x_1y_1 \cdots x_2y_2 \cdots b'b$ along the cycle O such that none of the vertices of the path except for a, b belongs to C_i, then if the rest of O is the path $as \cdots tb$, we replace the path $a'as \cdots tbb'$ by a path exactly as in case (i).

Suppose now that the edges e_1, e_2 belong to paths $aa' \cdots x_1y_1 \cdots b'b$ and $cc' \cdots x_2y_2 \cdots d'd$, along the cycle O such that none of the vertices of the paths except for a, b, c, d belongs to C_i. As above, not both a' and b' and not both c' and d' belong to R. Suppose without loss of generality that $a', d' \in N[v] - R$; then, $a'p_i, d'p_i \in E(\widehat{G})$

(Property P1). On the other hand, let $\widehat{b}=b$ if $b'\in R$ (then, $\widehat{b}\in C_i\cap S$) or $\widehat{b}=q_i$ otherwise, and let us define $\widehat{c}$ similarly. Then, we replace the path $a'a\cdots dd'$ by $a'p_id'$, and the path $b'b\cdots cc'$ by $b'\widehat{b}c'$ if $\widehat{b}=\widehat{c}$ or by $b'\widehat{b}q_i\widehat{c}c'$ if $\widehat{b}\neq\widehat{c}$. Note that if $e_1, e_2\notin\{aa', bb', cc', dd'\}$, they are not modified by the processing of C_i; if any coincides with aa', bb', cc', or dd', it is replaced by p_ia', $\widehat{b}b'$, $\widehat{c}c'$, and p_id', respectively, in accordance with the scheme C1.

After all the non-trivial co-components C_i have been processed, the above replacements yield a simple cycle in $\widehat{G}$ that passes through the edges $\widehat{e}_1, \widehat{e}_2$; this implies that $\widehat{e}_1, \widehat{e}_2$ belong to the same biconnected component of $\widehat{G}$, as desired.

($\Longleftarrow$) Let $\widehat{e}_1, \widehat{e}_2\in E(\widehat{G})$ be two edges belonging to the same biconnected component of $\widehat{G}$; we will show that the corresponding edges $e_1, e_2\in E(\overline{G})$ belong to the same biconnected component of $\overline{G}$. This half of the proof proceeds similarly to the first half. Because $\widehat{e}_1, \widehat{e}_2$ belong to the same biconnected component of $\widehat{G}$, there exists a simple cycle, say, $\widehat{O}$, in $\widehat{G}$ that passes through them. As above, we will replace parts of the cycle $\widehat{O}$ by paths in the graph $\overline{G}$, obtaining either a simple cycle in $\overline{G}$ that passes through e_1 and e_2, or a collection $O_1, O_2, \ldots, O_t$ of simple cycles in $\overline{G}$such that $e_1\in O_1$, $e_2\in O_t$, and for all $i=1, 2, \ldots, t-1$, there exist edges $e\in O_i$ and $e'\in O_{i+1}$ belonging to the same biconnected component of $\overline{G}$; in either of these cases, we conclude that the edges e_1, e_2 belong to the same biconnected component of $\overline{G}$.

Since all the vertices in $N[v]$ as well as those in $M(v)$ forming trivial co-components of the subgraph $G[M(v)]$ appear in both $\overline{G}$ and $\widehat{G}$, we need only concentrate in the parts of the cycle $\widehat{O}$ containing vertices that belong to non-trivial co-components of $G[M(v)]$.

Therefore, let us consider a non-trivial co-component C_i of the subgraph $G[M(v)]$. Then, the following property holds:

P2. Let $\widehat{a}a'$ be an edge of the graph $\widehat{G}$ such that $\widehat{a}\in\{p_i, q_i\}\cup(C_i\cap S)$ and $a'\notin\{p_i, q_i\}\cup(C_i\cap S)$, for a non-trivial co-component C_i of $G[M(v)]$; then, clearly $a'\in N[v]$. Moreover, there exists a vertex a such that $aa'\in E(\overline{G})$; in particular:

- if $a'\in R$, then $\widehat{a}\in C_i\cap S$ and $a'\widehat{a}\in E(\overline{G})$, i.e., $a=\widehat{a}$;
- if $a'\in N[v]-R$, then there exists a vertex $a\in C_i$ such that $aa'\in E(\overline{G})$ and the edge aa' belongs to the same biconnected component of $\overline{G}$ as all the edges of the subgraph $\overline{G}[\{v\}\cup C_i]$.

Property P2 is established in the same way as Property P1; the fact that if $a'\in N[v]-R$, the edge aa' belongs to the same biconnected component of $\overline{G}$ as all the edges of $\overline{G}[\{v\}\cup C_i]$ follows from Lemma 3.1(iii). Since all the edges of the subgraph $\widehat{G}[\{p_i, q_i\}\cup(C_i\cap S)]$ belong to the same biconnected component of $\widehat{G}$ (Lemma 3.3), and the algorithm correctly puts all the edges of the subgraph $\overline{G}[\{v\}\cup C_i]$ in the same biconnected component of $\overline{G}$ (Lemma 3.3 and Step 6), we do not need to consider edges $\widehat{e}_1, \widehat{e}_2$ that have all their four endpoints in $\{p_i, q_i\}\cup(C_i\cap S)$. Thus, we distinguish the following cases:

(i) *Exactly one of the edges $\widehat{e}_1, \widehat{e}_2$ has both its endpoints in $\{p_iq_i\}\cup(C_i\cap S)$:* Suppose that this edge is $\widehat{e}_1$, and let $\widehat{a}a'\cdots x_2y_2\cdots b'\widehat{b}$ be the subpath of $\widehat{O}$ passing through $\widehat{e}_2=x_2y_2$ such that only $\widehat{a}$, $\widehat{b}$ belong to C_i (note that the edge $\widehat{e}_2$ may coincide with the edges $\widehat{a}a'$ or $\widehat{b}b'$). According to Property P2, $a', b'\in N[v]$ and there exist vertices $a, b\in C_i$ such that $aa', bb'\in E(\overline{G})$. Additionally, it is important to note that not both a' and b' belong to the set R, for otherwise $\widehat{a}=\widehat{b}$, in contradiction to the fact that the cycle $\widehat{O}$ is simple and passes through the edge e_1 which has both endpoints in $\{p_i, q_i\}\cup(C_i\cap S)$. Thus, assume without loss of generality that $a'\in N[v]-R$. Then, if $a\neq b$, we replace the part $a'\widehat{a}\cdots x_1y_1\cdots\widehat{b}b'$ of the cycle passing through the edge $e_1=x_1y_1$ by the path $a'\rho b'$, where ρ is any a–b path in $\overline{G}$ (such a path ρ always exists as $\overline{G}[C_i]$ is connected), and the edge e_1 is any edge of the path ρ. Suppose now that $a=b$, and let $w\in M(v)-\{a\}$ be a vertex non-adjacent to a' in G; recall that $a'\in N[v]-R$). Then, we replace the cycle $\widehat{O}$ by the cycle $aa'\cdots x_2y_2\cdots b'$ and we add the cycle $O_1=vaa'w$; the two cycles share edge aa', and we consider as edge e_1 the edge va. In either case, the edge e_1 follows our correspondence scheme C2; in turn, if $\widehat{e}_2$ differs from both aa' and bb', then it is not modified, whereas otherwise it is replaced by aa' or bb', again in accordance with the scheme C2 holds.

(ii) *None of the edges $\widehat{e}_1, \widehat{e}_2$ has both its endpoints in $\{p_iq_i\}\cup(C_i\cap S)$:* If the edges $\widehat{e}_1=x_1y_1$ and $\widehat{e}_2=x_2y_2$ belong to a path $\widehat{a}a'\cdots x_1y_1\cdots x_2y_2\cdots b'\widehat{b}$ of the cycle $\widehat{O}$ such that none of the vertices of the path except for $\widehat{a}$, $\widehat{b}$ belongs to C_i, then we find vertices $a, b\in C_i$ such that $a'a, b'b\in E(\overline{G})$ (Property P2) and we replace $\widehat{O}$ by

the cycle $a' \cdots x_1 y_1 \cdots x_2 y_2 \cdots b' \rho$, where ρ is a b–a path in $\overline{G}[C_i]$; note that if $a', b' \in R$, the edges $\widehat{a}a', \widehat{b}b'$ are not modified, whereas if $a', b' \in N[v] - R$, either they do not change or they are replaced by edges $a'x, b'y$ where $x, y \in C_i$, in accordance with our scheme C2.

(ii) Suppose now that the edges $\widehat{e}_1, \widehat{e}_2$ belong to paths $\widehat{a}a' \cdots x_1 y_1 \cdots b'\widehat{b}$ and $\widehat{c}c' \cdots x_2 y_2 \cdots d'\widehat{d}$ of $\widehat{O}$ such that none of the vertices of the paths except for $\widehat{a}, \widehat{b}, \widehat{c}, \widehat{d}$ belongs to C_i. As discussed in case (i), there exists vertices $a, b, c, d \in C_i$ such that $aa', bb', cc', dd' \in E(\overline{G})$, and not both a' and b', and similarly c' and d' belong to R. Assume without loss of generality that $a', d' \in N[v] - R$; then, the edges aa' and dd' belong to the same biconnected component of $\overline{G}$ (Lemma 3.1(iii). Then, we find a b–a path ρ and a d–c path ρ' in the subgraph $\overline{G}[C_i]$ (such paths always exist as $\overline{G}[C_i]$ is connected), and we get the cycles $a' \cdots x_1 y_1 \cdots b' \rho$ and $c' \cdots x_2 y_2 \cdots d' \rho'$, where the edges aa' and dd' belong to the same biconnected component of $\overline{G}$. Note that the edges $\widehat{a}a', \widehat{b}b', \widehat{c}c', \widehat{d}d'$ are replaced by aa', bb', cc', dd', where $a, b, c, d \in C_i$ in accordance with our correspondence scheme C2.

After all the non-trivial co-components of $G[M(v)]$ that contributed vertices to the cycle $\widehat{O}$ have been processed, the above replacements yield either a simple cycle in $\overline{G}$ that passes through e_1 and e_2, or a collection $O_1, O_2, \ldots, O_t$ of simple cycles in $\overline{G}$ such that $e_1 \in O_1$, $e_2 \in O_t$, and for all $i = 1, 2, \ldots, t-1$, there exist edges $e \in O_i$ and $e' \in O_{i+1}$ belonging to the same biconnected component of $\overline{G}$, as desired. □

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 numbers of vertices and edges of G, recall that the degree of the vertex v is $|N(v)| = \mathrm{O}(\sqrt{m})$ (Lemma 2.1) and that the number k of co-components of the subgraph $G[M(v)]$ is $\mathrm{O}(\sqrt{m})$ (Lemma 3.1(ii)). Then, the following facts hold:

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

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

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

F4. *The sum of the cardinalities of the edge sets $\mathscr{B}_1, \mathscr{B}_2, \ldots, \mathscr{B}_\ell$* is $\mathrm{O}(n+m)$. This 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 $\widehat{\mathscr{B}}_j$ (Lemma3.3); then, each vertex $w \in C_i - S$ contributes exactly the edge vw in exactly one set $\mathscr{B}_j$.

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 of size n, and the sets $\widehat{\mathscr{B}}_i$ and $\mathscr{B}_i$, $i = 1, 2, \ldots, \ell$, are stored in linked lists.

Step 1 clearly takes $\mathrm{O}(n+m)$ time and $\mathrm{O}(n)$ space. Step 2 takes $\mathrm{O}(n+m)$ time and space since the co-components of a graph can be computed in time linear in its size [9,10,15,6]. In Step 3, computing the value of μ_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 $\mathrm{O}(|M(v)|) = \mathrm{O}(d(u))$ time as well. Thus, the execution of Step 3 for all the vertices in $N(v)$ takes $O(n + \sum_u d(u)) = \mathrm{O}(n+m)$ 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 $\mathrm{O}(n+m)$ time using $\mathrm{O}(n+m)$ space. Since the co-components $C_1, C_2, \ldots, C_k$ can be computed in $\mathrm{O}(n+m)$ time and space, and the `for`-loop in Step 4 takes $\mathrm{O}(\sqrt{m})$ time, the entire Step 4 is completed in $\mathrm{O}(n+m)$ 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 $\mathrm{O}(n+m)$ time and space. The pairs of adjacent vertices contained in the sets E_2 and E_3 can be easily determined and recorded in $\mathrm{O}(k) = \mathrm{O}(\sqrt{m})$ and $\mathrm{O}(n)$ time and space, respectively, while the pairs in E_4 can be found if, for each vertex $u \in N(v) - R$, we count its neighbors (in G) in each of the sets C_i, which can also be done in $\mathrm{O}(n+m)$ time and space. Thus, Step 5 can be carried out in $\mathrm{O}(n+m)$ 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 $\widehat{\mathscr{B}}_j$ (Lemma 3.3), and that the sum of the cardinalities of the sets $\widehat{\mathscr{B}}_j$ is $\mathrm{O}(m)$ (Fact F3). Finally, in light of Fact F4, Step 7 takes $\mathrm{O}(n+m)$ time. Summarizing, we have:

Theorem 3.1. *Let G be an undirected graph on n vertices and m edges. Then, 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. 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

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

in Step 5 of the sequential algorithm by the edge set

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

Note that $E_4 \subseteq E'_4$, since if for a vertex $u \in N(v) - R$ there exists $y \in C_i$ such that $uy \notin E(G)$, then u and all the vertices in C_i belong to the same co-component of $G[(N(v) - R) \cup M(v)]$. On the other hand, E'_4 may contain edges up_i, uq_i incident on a vertex $u \in N(v) - R$ although u is adjacent in G to all the vertices in C_i. It is as if we apply the algorithm on a graph G' which is obtained from G after the removal of some edges uw where $u \in N(v) - R$ and $w \in M(v)$; equivalently, the complement of G' is obtained from the complement $\overline{G}$ after the addition of these edges. Interestingly, the condition that "u and the representative of C_i belong to the same co-component of $G[(N(v) - R) \cup M(v)]$" ensures that this change does not affect the correctness of the algorithm because, as we show in Lemma 3.6, the endpoints of these added edges belong to the vertex set of the same biconnected component of $\overline{G}$ (see Lemma 2.4(i)).

Lemma 3.6. *Let G be an undirected graph, v a vertex of G, $M(v)$ the non-neighbors of v in G, R the set of neighbors of v with at most one non-neighbor in $M(v)$, and $\widetilde{G}$ the subgraph $G[(N(v) - R) \cup M(v)]$. 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 vertex set of 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 vertex set of the same biconnected component of $\overline{G}$; then, since for any three consecutive vertices p, q, r in a path in the complement of $\widetilde{G}$, the biconnected components to which p, q and q, r belong share the vertices q and v, the lemma follows from Lemma 2.4(iv).

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 vertex set of the same biconnected component of $\overline{G}$. The same conclusion is also reached if $x \in N(v) - R$ and $y \in M(v)$ as a result of Lemma 3.1(iii). Now, suppose that $x, y \in N(v) - R$, and let $w, w', w'' \in M(v)$ be vertices such that $xw, yw', yw'' \in E(\overline{G})$. Then, x, y, v, w, w' all belong to the vertex set of the same biconnected component of $\overline{G}$: this follows from the simple cycle $xyw''vw$ in $\overline{G}$ if $w = w'$, and from the simple cycle $xyw'vw$ in $\overline{G}$ if $w \neq w'$. □

Next, we analyze the time and processor complexity of each step of the algorithm on the PRAM model of parallel computation; for details on the PRAM techniques mentioned below, see [1,16]. 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); all these links can be easily established in optimal $O(1)$ time using $O(m)$ processors on the EREW PRAM model using an auxiliary array [16].

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 $d(v)$ denotes the degree of vertex v 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 a 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: If the degree of the vertex v is equal to 0 in G, then the algorithm computes the connected co-components $C_1, C_2, \ldots, C_t$ of the graph $G[V(G)-\{v\}]$. This computation is done by applying the optimal parallel co-connectivity algorithm of Chong et al. [6] which, on the graph $G[V(G)-\{v\}]$ on $n-1$ vertices and $O(m)$ 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 μ_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; this is done by computing the degree of w in the subgraph $G[R' \cup M(v)]$ induced by the vertices in R' and $M(v)$. Then, if $a_w < |R'|$ we insert the vertex w in S. The construction of the subgraph $G[R' \cup M(v)]$ can be done in $O(\log n)$ time using $O((n+m)/\log n)$ processors on the EREW PRAM model, while the computation of the values a_w can be carried out within the same time and processor complexity as the computation of the vertex degrees in G (see Step 1) and along with the updating of S take $O(\log n)$ 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 a copy of the adjacency-list representation 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 $k \leqslant \sqrt{2m} = O(n)$.

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 $|S| = O(\sqrt{m})$ (Fact F1), 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 $|N(v) - R| = O(|N(v)|) = O(\sqrt{m})$ and the number k of co-components C_i is $O(\sqrt{m})$, 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 $\widehat{G}$ 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 λ be the total number of edges in the biconnected components $\widehat{\mathcal{B}}_1, \widehat{\mathcal{B}}_2, \ldots, \widehat{\mathcal{B}}_\ell$; Fact F3 implies that $\lambda = O(m)$. The algorithm which computes $\widehat{\mathcal{B}}_1, \widehat{\mathcal{B}}_2, \ldots, \widehat{\mathcal{B}}_\ell$ generates an output array $b[\,]$ of length λ such that $b[e]$ is equal to an edge which is the representative of the biconnected component containing edge e; let $e_1, e_2, \ldots, e_\lambda$ be the edges of the biconnected components and let $\widehat{e}_1, \widehat{e}_2, \ldots, \widehat{e}_\ell$ be the representatives.

Let γ be the number of vertices in the set $\bigcup_{i=1}^{k}(C_i - S)$, and let these vertices be $w_1, w_2, \ldots, w_\gamma$. Let $e'_1, e'_2, \ldots, e'_\gamma$ be the edges $vw_1, vw_2, \ldots, vw_\gamma$ of the complement $\overline{G}$. We construct an array $c[\,]$ of length γ; the entry $c[e'_i]$, for each edge $e'_i = vw_i$, is initially assigned the dummy value e^*. For each edge p_iq_i, $1 \leqslant i \leqslant k$, we find the representative $\widehat{e}_j$ of the biconnected component $\widehat{\mathscr{B}}_j$ to which p_iq_i belongs (recall that $b[p_iq_i] = \widehat{e}_j$); this computation can be done in $O(\log n)$ time using $O(m/\log n)$ processors on the EREW PRAM model. Then, for each vertex $w_j \in C_i - S$, we assign to $c[vw_j]$ the representative $\widehat{e}_t$ of the biconnected component of $\widehat{G}$ to which the edge p_iq_i belongs, that is, $c[vw_j] = \widehat{e}_t$; 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 representative $\widehat{e}_t$, which for all C_is can be done in $O(\log n)$ time using $O(n/\log n)$ processors on the same model of computation). Finally, in order to remove from $\widehat{\mathscr{B}}_1, \widehat{\mathscr{B}}_2, \ldots, \widehat{\mathscr{B}}_\ell$ all edges e in $\{e_1, e_2, \ldots, e_\lambda\}$ that are incident on some p_i or q_i for $1 \leqslant i \leqslant k$ (such edges correspond to entries of the array $b[\,]$ only), we assign to $b[e]$ the value e^* if at least one of the endpoints of e is some p_i or q_i. 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 $\lambda + \gamma$ such that $d[i] = (e_i, b[e_i])$, $1 \leqslant i \leqslant \lambda$, and $d[\lambda + j] = (e'_j, c[e'_j])$, $1 \leqslant j \leqslant \gamma$. Then, we delete all the entries of $d[\,]$ of the form (e, e^*) and move the remaining entries in consecutive locations; the resulting array represents the elements of the edge sets $\mathscr{B}_1, \mathscr{B}_2, \ldots, \mathscr{B}_\ell$. 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 $\mathscr{B}_1, \mathscr{B}_2, \ldots, \mathscr{B}_\ell$ 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, we obtain the following theorem.

Theorem 3.2. *Let G be an undirected graph on n vertices and m edges. Then, algorithm* BICONNECTED_CO-COMPONENTS *can be parallelized to yield the biconnected components of $\overline{G}$ 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 $\widehat{G}$ in Step 6 (due to the use of the CREW PRAM algorithm in [16]), 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 co-components

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. As with the case of the biconnected components of the complement of an undirected graph, the algorithm uses an auxiliary graph of small size which, however, captures all the information on the strongly connected components of the complement of the given directed graph G. More specifically, for a vertex v of G of minimum sum of in-degree and out-degree, we define the directed graph H_v as follows:

$$
\begin{aligned}
V(H_v) &= N(v) \cup \{v\} = N^-(v) \cup N^+(v) \cup \{v\},\\
E(H_v) &= \{xy \mid x, y \in N(v) \text{ and } xy \in E(G)\} \cup \{vx \mid x \in N(v) \text{ and } M(v) \cup \{v\} \subseteq N^-(x)\}\\
&\quad \cup \{xv \mid x \in N(v) \text{ and } M(v) \cup \{v\} \subseteq N^+(x)\},
\end{aligned}
$$

where $M(v) = V(G) - (N(v) \cup \{v\})$ is the set of non-neighbors of v in G. The graph H_v has as vertices the vertex v and all its neighbors in G, i.e., all the vertices that are adjacent to or from v in G; its edge set contains all the edges between neighbors of v in G, edges vx from only the vertices x that are adjacent to v and to all the non-neighbors of v in G, and edges xv from only the vertices x that are adjacent from v and from all the non-neighbors of v in G. Fig. 2 shows a directed graph G and the corresponding graph H_v. The usefulness of the graph H_v is shown in the following lemma.

Lemma 4.1. *Let G be a directed graph, v a vertex of G, $N(v) = N^-(v) \cup N^+(v)$ the set of neighbors of v in G, and $M(v) = V(G) - (N(v) \cup \{v\})$ the set of non-neighbors of v.*

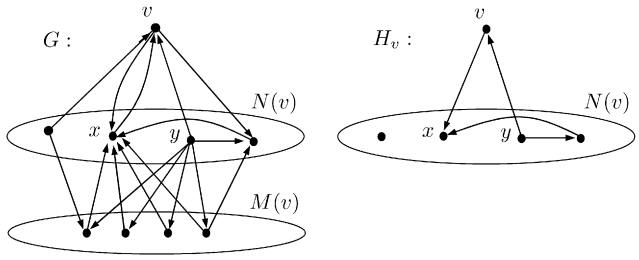

Fig. 2. A directed graph G and the corresponding directed graph H_v.

(i) *If x is a vertex of G such that $x \notin N(v)$, i.e., none of the directed edges vx and xv belongs 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 defined above. Then, any two vertices $x, y \in V(H_v)$ belong to the same strongly connected component of $\overline{G}$ if and only if they belong to the same strongly connected component of $\overline{H}_v$.*

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

(ii) ($\Longrightarrow$) Suppose that the vertices $x, y \in V(H_v)$ 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 $x-y$ path and a directed $y-x$ path; 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 shortest such paths, respectively. We distinguish the following two cases:

(a) $x = v$ and $y \in N(v)$: If $u_i \in N(v)$ for all $i = 1, 2, \ldots, k$, then the path $vu_1\cdots u_{k-1}y$ is also a v–y path in $\overline{H}_v$. Otherwise, since $u_0u_1\cdots u_k$ is a shortest v–y path in $\overline{G}$, we conclude that $u_1 \in M(v)$ and $u_i \in N(v)$ for $2 \leqslant i \leqslant k$. Moreover, because $u_1u_2 \in E(\overline{G})$ where $u_1 \in M(v)$ and $u_2 \in N(v)$, we have that $M(v) \cup \{v\} \nsubseteq N^-(u_2)$ 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 $w_i \in N(v)$ for all $i = 0, 1, \ldots, \ell-1$, then the path $yw_1\cdots w_{\ell-1}v$ is also a y–v path in $\overline{H}_v$; otherwise, $w_{\ell-1} \in M(v)$ and $w_i \in N(v)$ for $0 \leqslant i \leqslant \ell-2$, 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) $x, y \in N(v)$: If $u_i \in N(v)$ for all $i = 0, 1, \ldots, k$, then the path $xu_1\cdots u_{k-1}y$ is also an $x-y$ path in $\overline{H}_v$. Otherwise, let $k' = \min_i\{u_i \in M(v) \cup \{v\}\}$; k' is well defined (since there exists a vertex $u_i \notin N(v)$), $k' > 0$, and $u_i \in N(v)$ for $0 \leqslant i < k'$; moreover, because $u_{k'-1}u_{k'} \in E(\overline{G})$, i.e., $u_{k'-1}u_{k'} \notin E(G)$, and because $u_{k'-1} \in N(v)$ and $u_{k'} \in M(v) \cup \{v\}$, we have that $u_{k'-1}v \notin E(H_v)$ or equivalently $u_{k'-1}v \in E(\overline{H}_v)$. Additionally, if $k'' = \max_i\{u_i \in M(v) \cup \{v\}\}$, then k'' is well defined, $k'' < k$, $u_i \in N(v)$ for $k'' < i \leqslant k$, 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 $w_i \in N(v)$ for all $i = 0, 1, \ldots, \ell$, then the path $yw_1\cdots w_{\ell-1}x$ is also a $y-x$ path in $\overline{H}_v$; otherwise, if $\ell' = \min_i\{w_i \in M(v) \cup \{v\}\}$ and $\ell'' = \max_i\{w_i \in M(v) \cup \{v\}\}$, 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 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 $y \in N(v)$: Because the above paths are shortest, then $u_i \in N(v)$ for all $i = 1, 2, \ldots, k$, and $w_j \in N(v)$ for all $j = 0, 1, \ldots, \ell-1$. Since $vu_1 \in E(\overline{H}_v)$, then $M(v) \cup \{v\} \nsubseteq N^-(u_1)$, i.e., there exists a vertex $z \in M(v) \cup \{v\}$ such that $zu_1 \in E(\overline{G})$. Similarly, since $w_{\ell-1}v \in E(\overline{H}_v)$, there exists a vertex $z' \in M(v) \cup \{v\}$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) $x, y \in N(v)$: If $u_i \in N(v)$ for all $i=0, 1, \ldots, k$, 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, where $1 \leqslant j \leqslant k-1$. Then, $u_i \neq v, \forall i \neq j$. Since $u_{j-1}v \in E(\overline{H}_v)$, then $M(v) \cup \{v\} \not\subseteq N^+(u_{j-1})$, i.e., there exists a vertex $a \in M(v) \cup \{v\}$ such that $u_{j-1}a \in E(\overline{G})$. Similarly, since $vu_{j+1} \in E(\overline{H}_v)$, there exists a vertex $b \in M(v) \cup \{v\}$ 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 $w_i \in N(v), \forall i = 0, 1, \ldots, \ell$) or there exist an index j' $(1 \leqslant j' \leqslant \ell - 1)$ and vertices $a', b' \in M(v) \cup \{v\}$ 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 minimum 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 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 earlier 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 both its in-degree and its out-degree 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 numbers 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, 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 out-degree $d^+(u)$ of a vertex u of the graph G can be easily computed 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. The computation for all the vertices takes $O(\log n)$ time and $O(m/\log n)$ processors on the same model of computation. The computation of the in-degree of a vertex u can be done 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: We first construct the subgraph G_v of the graph G induced by the vertex set $N(v) \cup \{v\}$. An adjacency-list representation of the subgraph G_v can be obtained by first processing a copy of the adjacency-list representation of G, and re-indexing (based on the ranks of the vertices in the adjacency lists of G) so that the vertices in $N(v) \cup \{v\}$ 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 x 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 edge 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 not difficult 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 $|N(v) \cup \{v\}| = O(\sqrt{m})$ (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 $N(v) \cup \{v\}$; since $|N(v) \cup \{v\}| = O(\sqrt{m})$, 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, algorithm* STRONG_CO-COMPONENTS *can be parallelized to yield the strongly connected components of $\overline{G}$ 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 connected co-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. In this case, for a vertex v of the input graph G, we define the following auxiliary undirected graph H_v:

$$V(H_v) = N[v],$$
$$E(H_v) = \{xy \mid x, y \in N(v) \text{ and } xy \in E(G)\} \cup \{vx \mid x \in N(v) \text{ and } M(v) \cup \{v\} \subseteq N(x)\},$$

where $N(v)$ and $M(v)$ denote the sets of neighbors and non-neighbors of v in G. 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 $M(v)$ be the sets of neighbors and non-neighbors of v* in *G.*

(i) *If x is a non-neighbor of v in G, then the vertices v and x belong to the same connected component of $\overline{G}$.*
(ii) *Let H_v be the undirected graph defined earlier. Then, two vertices $x, y \in N(v) \cup \{v\}$ belong to the same connected component of $\overline{G}$ if and only if they belong to the same connected component of $\overline{H}_v$.*

Then, an algorithm similar to STRONG_CO-COMPONENTS applied on a vertex v of minimum degree in G yields a very simple optimal sequential algorithm for computing the connected components of the complement of the 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 described 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 obtained a simple optimal $O(\log n)$-time EREW PRAM co-connectivity algorithm.

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 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.

References

[1] S.G. Akl, Parallel Computation: Models and Methods, Prentice-Hall, Englewood Cliffs, NJ, 1997.
[2] B. Awerbuch, Y. Shiloach, New connectivity and MSF algorithms for ultra-computer and PRAM, IEEE Trans. Comput. 36 (1987) 1258–1263.
[3] F.Y. Chin, J. Lam, I. Chen, Efficient parallel algorithms for some graph problems, Comm. ACM 25 (1982) 659–665.
[4] K.W. Chong, Y. Han, Y. Igarashi, T.W. Lam, Improving the efficiency of parallel minimum spanning tree algorithms, Discrete Appl. Math. 126 (2003) 33–54.
[5] K.W. Chong, Y. Han, T.W. Lam, Concurrent threads and optimal parallel minimum spanning trees algorithm, J. ACM 48 (2001) 297–323.
[6] K.W. Chong, S.D. Nikolopoulos, L. Palios, An optimal parallel co-connectivity algorithm, Theory Comput. Syst. 37 (2004) 527–546.
[7] R. Cole, U. Vishkin, Faster optimal prefix sums and list ranking, Inform. and Comput. 81 (1989) 334–352.
[8] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, second ed., MIT Press, Inc., Cambridge, MA, 2001.
[9] E. Dahlhaus, J. Gustedt, R.M. McConnell, Efficient and practical algorithms for sequential modular decomposition, J. Algorithms 41 (2001) 360–387.
[10] E. Dahlhaus, J. Gustedt, R.M. McConnell, Partially complemented representation of digraphs, Discrete Math. Theoret. Comput. Sci. 5 (2002) 147–168.
[11] H. Gazit, G.L. Miller, An improved parallel algorithm that computes the BFS numbering of a directed graph, Inform. Process. Letters 28 (1988) 61–65.
[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, in: Proceedings of the Eighth ACM Symposium on Theory of Computing (STOC'76), 1976, pp. 55–57.
[14] D.S. Hirschberg, A.K. Chandra, D.V. Sarwate, Computing connected components on parallel computers, Comm. ACM 22 (1979) 461–464.
[15] H. Ito, M. Yokoyama, Linear time algorithms for graph search and connectivity determination on complement graphs, Inform. Process. Letters 66 (1998) 209–213.
[16] J. JáJá, An Introduction to Parallel Algorithms, Addison-Wesley, Reading, MA, 1992.
[17] M.-Y. Kao, Linear-processor NC algorithms for planar directed graphs I: strongly connected components, SIAM J. Comput. 22 (1993) 431–459.
[18] D. Nath, S.N. Maheshwari, Parallel algorithms for the connected components and minimal spanning trees, Inform. Process. Letters 14 (1982) 7–11.
[19] J. Reif, Depth-first search is inherently sequential, Inform. Process. Letters 20 (1985) 229–234.
[20] J. Reif (Ed.), Synthesis of Parallel Algorithms, Morgan Kaufmann Publishers, San Mateo, California, 1993.
[21] C. Savage, J. JáJá, Fast efficient parallel algorithms for some graph problems, SIAM J. Comput. 10 (1981) 682–691.
[22] Y. Shiloach, U. Vishkin, An $O(\log n)$ parallel connectivity algorithm, J. Algorithms 3 (1982) 57–67.
[23] R. Tarjan, U. Vishkin, Finding biconnected components and computing tree functions in logarithmic parallel time, SIAM J. Comput. 14 (1985) 862–874.
[24] Y.H. Tsin, F.Y. Chin, Efficient parallel algorithms for a class of graph theoretic problems, SIAM J. Comput. 13 (1984) 580–599.