A simple certifying algorithm for 3-edge-connectivity

Yung H. Tsin*

Abstract

A linear-time certifying algorithm for 3-edge-connectivity is presented. Given a connected undirected graph $G$, if $G$ is 3-edge-connected, the algorithm generates a construction sequence as a positive certificate for $G$. Otherwise, the algorithm decomposes $G$ into its 3-edge-connected components and generates a construction sequence for each of them as well as the bridges and a cactus representation of the cut-pairs in $G$ as negative certificates. All of these are done by making only one pass over $G$ using an innovative graph contraction technique. Moreover, the graph need not be 2-edge-connected. The currently best-known algorithm is more complicated as it makes multiple passes over $G$ and uses involved reduction and perturbation techniques rather than just basic graph-theoretic techniques.

Keywords: certifying algorithm, graph recognition algorithm, 3-edge-connected graph, 3-edge-connected component, Mader construction sequence, cactus representation of cut-pair, certificate, depth-first search.

*School of Computer Science, University of Windsor, Windsor, Ontario, Canada, N9B 3P4; peter@uwindsor.ca. Research supported by U of Windsor, GRF grant #815160.
1 Introduction

A major problem in software engineering is correctness of software. Even after an algorithm is proven correct by its designer, its implementation as a program may still contain bugs (implementation errors). This is particularly true for complex algorithms as their implementations tend to be error-prone. Consequently, it is difficult to tell if an output generated by a program is correct or has been compromised by a bug.

McConnell et al. [16] addressed this problem by introducing certifying algorithms. A certifying algorithm is an algorithm that generates a certificate along with its output for each input. The certificate is an evidence that can be used by an authentication algorithm to verify the correctness of the output. An authentication algorithm is a separate algorithm that takes the input, the output, and the certificate to verify (independently of the algorithm) whether the output is correct. Certifying algorithms are of great value in practice as the user can verify the correctness of the output they received for each input regardless of whether the program is bug-free. Certifying algorithms have been extensively used in the software library LEDA [17] after a user discovered an error in a program for testing graph planarity. The inclusion of these algorithms greatly improved the reliability of the library.

A graph recognition problem is a graph-theoretic problem that checks for membership of a class of graphs. A certifying algorithm for a graph recognition problem generates a positive certificate if the answer is ‘yes’ and a negative certificate if the answer is ‘no’. Efficient recognition algorithms exist for many graph recognition problems [4, 6, 11, 12, 13, 19, 22, 26]. However, only a few are certifying.

In this paper, we address the 3-edge-connectivity problem which has applications in a wide variety of areas [1, 2, 3, 5, 14, 25]. A number of linear-time algorithms had been proposed [9, 20, 28, 30, 31]. However, none of them is certifying. Recently, Mehlhorn, Neumann and Schmidt [19] presented a linear-time certifying algorithm for testing 3-edge-connectivity. Their algorithm returns a Mader construction sequence as a positive certificate if the graph is 3-edge-connected and returns a cut-pair as a negative certificate otherwise. The key idea underlying their algorithm is as follows.

Given a 2-edge-connected undirected graph $G = (V, E)$, the algorithm first makes one pass over $G$ to decompose $G$ into a collection of chains, $C_1, C_2, \ldots, C_{|E|-|V|+1}$, which are cycles or paths. Then, starting from the graph $G_c$ which is a $K^3_2$-subdivision consisting of $C_1$ and $C_2$, the remaining chains are added one at a time to $G_c$ to generate a Mader construction sequence for $G$. The chains are classified as interlacing or nested. An interlacing chain can be added immediately to the current $G_c$ if its terminating vertices are in $G_c$. Each nested chain is combined with a group of chains to form a segment so that once the chain is
added to the current $G_c$, the remaining chains in the segment become interlacing and can also be added. The algorithm proceeds in phases. In Phase $i$, $1 \leq i \leq |E| - |V| + 1$, the set of chains whose starting end-vertex is on $C_i$ are determined and all the interlacing chains that can be added into the current $G_c$ are added. For the segments that are determined by nested chains, a correct order to add them to the current $G_c$ is to be determined. This is achieved by reducing the problem to a problem on intervals based on the notion of overlap graph as follows:

First an overlap graph based on the segments is generated. Then by associating each segment with a set of intervals, a graph on the intervals is constructed. Then, using a method of Olariu and Zomaya [24] and a perturbation technique for handling intervals having common endpoint, a spanning forest in the graph is constructed. The ordering exists and hence the algorithm can carry on if the spanning forest is a spanning tree. Otherwise, the algorithm terminates execution and outputs a cut-pair as a negative certificate that the given graph is not 3-edge-connected. Mehlhorn et al. also explained how to extend their algorithm to generate a cactus representation of the cut-pairs if $G$ is not 3-edge-connected.

Although the ideas underlying the algorithm are elegant, the algorithm is rather complicated owing to the multiple passes it makes over the input graph and the non-trivial reduction and perturbation techniques it uses. For instance, the algorithm requires that the input graph is 2-edge-connected. Therefore, to use their algorithm, an application programmer must first implement a bridge-connected-component algorithm to decompose the input graph into a set of 2-edge-connected components and then runs the certifying algorithm on each of them. This would increase the workload of the programmer and induce substantial run-time overhead. The reduction technique also requires data structures to be created during run-time to represent the overlap graphs. Straightly speaking, Mehlhorn et al. only presented a certifying algorithm for testing if a 2-edge-connected graph is 3-edge-connected. It produces only a cut-pair if the answer is negative. Although they did explain how to extend the algorithm to generate a construction sequence for each 3-edge-connected-component and a cactus representation of the cut-pairs, no pseudo-codes are provided. Hence, it would not be easy for an application programmer to implement the extension correctly. For generating a construction sequence for each 3-edge-connected-component, it is likely that they first implement an existing non-certifying 3-edge-connected-component algorithm to generate the 3-edge-connected components of the input graph and then uses the certifying algorithm of Mehlhorn et al. to generate a construction sequence for each 3-edge-connected-component, making yet one more pass over the input graph.

In this paper, we present a simpler algorithm as an alternative. Specifically, we present a linear-time certifying algorithm that makes only one pass over the input graph $G$ (not necessarily 2-edge-connected) to
seamlessly generate all 3-edge-connected components of the graph each with a Mader construction sequence as well as all the bridges and a cactus representation of the cut-pairs. Clearly, if there is only one Mader construction sequence, \(G\) is 3-edge-connected with the construction sequence as a positive certificate; if there are more than one Mader construction sequences, \(G\) is not 3-edge-connected and the cactus representation or any cut-pair in it is a negative certificate. Moreover, the Mader construction sequences associated with each 3-edge-connected component serves as a positive certificate for that 3-edge-connected component. The algorithm is a simple extension of the 3-edge-connected-component algorithm of Tsin [30].

Since our algorithm is a one-pass algorithm and does not reduce any of the subtasks (generating 3-edge-connected components, generating Mader construction sequences, generating bridges, generating cactus representation of cut-pairs) to other computational problems, the various parts of our algorithm for solving the subtasks thus constitute a cohesive algorithm that is self-contained, conceptually simple, and easy to implement. Hence, it is more likely used by experimental scientists and application programmers.

2 Some basic definitions and facts

The definitions of the graph-theoretic concepts used in this article are standard and can be found in many textbooks or references such as [7, 29]. However, to make the article self-contained, we give some of the important definitions below.

An undirected graph is represented by \(G = (V, E)\), where \(V\) is the vertex set and \(E\) is the edge set. An edge \(e\) with \(u\) and \(v\) as end-vertices is represented by \(e = (u, v)\). The graph may contain parallel edges (two or more edges sharing the same pair of end vertices). The degree of a vertex \(u\) in \(G\), denoted by \(deg_G(u)\), is the number of edges with \(u\) as an end-vertex. A path \(P\) in \(G\) is a sequence of alternating vertices and edges, \(u_0e_1u_1e_2u_2\ldots e_ku_k\), such that \(u_i \in V, 0 \leq i \leq k, e_i = (u_{i-1}, u_i), 1 \leq i \leq k\), where \(u_i, 0 \leq i \leq k\), are distinct with the exception that \(u_0\) and \(u_k\) may be identical. The edges \(e_i, 1 \leq i \leq k\), could be omitted if no confusion could occur as a result. The path is a null path if \(k = 0\) and is a cycle if \(u_0 = u_k\). The path is called an \(u_0 - u_k\) path with vertices \(u_0\) and \(u_k\) as terminating vertices and \(u_i, 1 \leq i < k\), as internal vertices. If the path \(P\) is given an orientation from \(u_0\) to \(u_k\), then \(u_0\) is the source, denoted by \(s(P)\), and \(u_k\) is the sink, denoted by \(t(P)\), of \(P\) and the path \(P\) is also represented by \(u_0 \rightsquigarrow_G u_k\). The graph \(G\) is connected if \(\forall u, v \in V\), there is a \(u - v\) path in it. It is disconnected otherwise. Let \(G = (V, E)\) be a connected graph. An edge is a bridge in \(G\) if removing it from \(G\) results in a disconnected graph. The graph \(G\) is 2-edge-connected if it has no bridge. A cut-pair of \(G\) is a pair of edges whose removal results in a
disconnected graph and neither is a bridge. A cut-edge is an edge in a cut-pair. \( G \) is 3-edge-connected if it is bridgeless and has no cut-pair. A 3-edge-connected component (abbreviated 3ecc) of \( G \) is a maximal subset \( U \subseteq V \) such that \( \forall u, v \in U, u \neq v \), there exists three edge-disjoint \( u - v \) paths in \( G \). A graph \( G' = (V', E') \) is a subgraph of \( G \) if \( V' \subseteq V \) and \( E' \subseteq E \). Let \( U \subseteq V \), the subgraph of \( G \) induced by \( U \), denoted by \( G_{(U)} \), is the maximal subgraph of \( G \) whose vertex set is \( U \). Let \( D \subseteq E \), \( G \setminus D \) denotes the graph resulting from \( G \) after the edges in \( D \) are removed.

Depth-first search (abbreviated dfs) augmented with vertex labeling is a powerful graph traversal technique first introduced by Tarjan \[29\]. When a dfs is performed over a graph, each vertex \( w \) is assigned a depth-first number, \( dfs(w) \), such that \( dfs(w) = k \) if vertex \( w \) is the \( k \)th vertex visited by the search for the first time. The search also partitions the edge set into two types of edges, tree-edge and back-edge and gives each edge an orientation. With the orientation taken into consideration, a tree-edge \( e = (u, v) \) is denoted by \( u \to v \) where \( dfs(u) < dfs(v) \) and a back-edge \( e = (u, v) \) is denoted by \( v \prec u \), where \( dfs(v) < dfs(u) \).

In the former case, \( u \) is the parent of \( v \), denoted by \( u = parent(v) \), while \( v \) is a child of \( u \). In the latter case, \( (v \prec u) \) is an incoming back-edge of \( v \) and an outgoing back-edge of \( u \). In either case, \( u \) is the tail, denoted by \( s(e) \), while \( v \) is the head, denoted by \( t(e) \), of the edge. The tree edges form a directed spanning tree \( T = (V, E_T) \) of \( G \) rooted at the vertex \( r \) from which the search begins. A path from vertex \( u \) to vertex \( v \) in \( T \) is denoted by \( u \leadsto_T v \). Vertex \( u \) is an ancestor of vertex \( v \), denoted by \( u \preceq v \), if and only if \( u \) is a vertex on \( r \leadsto_T v \). Vertex \( u \) is a proper ancestor of \( v \), denoted by \( u \prec v \), if \( u \preceq v \) and \( u \neq v \). Vertex \( v \) is a (proper) descendant of vertex \( u \) if and only if vertex \( u \) is an (proper) ancestor of vertex \( v \). When a depth-first search reaches a vertex \( u \), vertex \( u \) is called the current vertex of the search. The subtree of \( T \) rooted at vertex \( w \), denoted by \( T_w \), is the subtree containing all the descendants of \( w \).

\[ \forall w \in V, \text{lowpt}(w) = \min \{ \{dfs(w)\} \cup \{dfs(u) \mid (u \prec w) \in E \setminus E_T\} \cup \{\text{lowpt}(u) \mid (w \to u) \in E_T\} \}. \]

A subdivision is an operation that replaces an edge \( (u, v) \) with a path \( uxv \), where \( x \) is a new vertex. A subdivision of a graph \( G \) is a graph resulting from applying zero or more subdivision operations on \( G \). The graph \( K^3_3 \) is the graph consisting of two vertices and three parallel edges.

### 3 A certifying algorithm for 3-edge-connectivity

Mehlhorn et al. \[19\] showed that a construction sequence, called Mader construction sequence, can be used as a positive certificate for 3-edge-connected graphs. It is based on the following generalization of Mader’s Theorem \[15\].
Theorem 3.1. Every subdivision of a non-trivial 3-edge-connected graph (and no other graph) can be constructed from a subdivision of a $K_3^2$ using the following three operations:

(i) adding a path connecting two branch vertices (vertices of degree at least three);
(ii) adding a path connecting a branch vertex and a non-branch vertex;
(iii) adding a path connecting two non-branch vertices lying on distinct links (maximal paths whose internal vertices are of degree two).

In each of the above three cases, the path is called a Mader path and its internal vertices are new vertices.

In this section, we show that the 3-edge-connected-component algorithm of [30] can be easily extended to generate, in addition to the 3-edge-connected components, a Mader construction sequence for each of them, and all the bridges as well as a cactus representation of the cut-pairs if the given graph is not 3-edge-connected. All these are generated seamlessly with one depth-first search over the given graph $G$.

As with Mehlhorn et al., our Mader paths are ears of an ear decomposition of the 2-edge-connected components of the input graph. We use a ear decomposition of 2-edge-connected graphs used in [32]. Mehlhorn et al. [19] uses a decomposition method of Schmidt [27].

Let $T = (V, E_T)$ be a depth-first search tree of a 2-edge-connected graph $G$. Using the depth-first search numbers of the vertices, we can rank the back-edges as follows.

**Definition:** Let $(q \rightleftharpoons p)$ and $(y \rightleftharpoons x)$ be two back-edges. Then $(q \rightleftharpoons p)$ is lexicographically smaller than $(y \rightleftharpoons x)$, denoted by $(q \rightleftharpoons p) \prec (y \rightleftharpoons x)$, if and only if

(i) $dfs(q) < dfs(y)$, or

(ii) $q = y$ and $dfs(p) < dfs(x)$ such that $p$ is not an ancestor of $x$, or

(iii) $q = y$ and $p$ is a descendant of $x$, or

(iv) $q = y$ and $p = x$ and $(q \rightleftharpoons p)$ is encountered before $(y \rightleftharpoons x)$ during the dfs. (parallel edges)

Since every tree-edge is the parent edge of a unique vertex $u$, every tree-edge can be represented by $(parent(u) \rightarrow u)$. Using the back-edges, we can partition the edges of $G$ into edge-disjoint paths such that every path contains exactly one back-edge as follows: for each tree-edge $(parent(u) \rightarrow u)$, we associate with it the back-edge $(y \rightleftharpoons x)$ with the lowest rank in lexicographical order such that $x$ is a descendant of $u$ while $y$ is a proper ancestor of $u$. The back-edge exists because $G$ is 2-edge-connected. It is easily verified that the back-edge $(y \rightleftharpoons x)$ and all the tree-edges associated with it form a path $yv_1 \cdots v_pv$ such that $v \rightsquigarrow_T x$. The path is called an ear and is denoted by $P_{y \rightleftharpoons x} : yv_1 \cdots v_pv$. Note that $P_{y \rightleftharpoons x}$ is given an orientation from $y$ to $v$; i.e. $P_{y \rightleftharpoons x}$ is an $y \rightsquigarrow_G v$ such that $s(P_{y \rightleftharpoons x}) = y$ and $t(P_{y \rightleftharpoons x}) = v$. Furthermore,
if \((y \bowtie x)\) has the rank \(i\) lexicographically, we shall also denote the ear by \(P_i : yxv_1 \cdots v_pv\). As a result, the ears can be ranked lexicographically as \(P_1, P_2, \ldots, P_{m-n+1}\) which is called an ear-decomposition of \(G\). An ear is non-trivial if it contains at least one tree-edge and is trivial otherwise. Note that for a back-edge \((y \bowtie v)\), \(s(y \bowtie v) = v\) and \(t(y \bowtie v) = y\) but when it is considered as a trivial ear \(P_{y \bowtie v}\), then \(s(P_{y \bowtie v}) = y\) and \(t(P_{y \bowtie v}) = v\).

**Lemma 3.2.** Let \(G = (V, E)\) be a 2-edge-connected graph. \(G\) has an ear-decomposition, \(P_1, P_2, \ldots, P_{|E| - |V| + 1}\) in which \(P_1\) is a cycle and \(P_i, 2 \leq i \leq |E| - |V| + 1\) is a path or a cycle of which each terminating vertex lies on some ear \(P_j, 1 \leq j < i\), and no internal vertex lies on any ear \(P_j, 1 \leq j < i\).

**Proof:** Immediate from the definition of \(P_i\) and the 2-edge-connectivity of \(G\). ■

Contrary to Mehlhorn et. al, our ear-decomposition is not generated explicitly. Instead, it is generated by labeling every edge \(e \in E\) with the back edge that determines the ear containing \(e\). This back edge, denoted by \(\text{ear}(e)\), is determined during the depth-first search based on the following recursive definition:

\[
\text{ear}(e) = \begin{cases} 
  e & \text{if } e \in E \backslash E_T; \\
  \min_\prec(\{f \mid f = (u \bowtie w) \in E \backslash E_T\} \cup \\
  \{\text{ear}(f) \mid f = (w \to u) \in E_T\}), & \text{if } e = (\text{parent}(w) \to w) \in E_T 
\end{cases}
\]

The ears are generated only when the construction sequence is constructed at the end.

In the sequel, let \(e = (v, w) \in E\), if \(e \in E_T\) \((e \in E \backslash E_T, \text{respectively})\), we shall use \(\text{ear}((v, w))\) and \(\text{ear}(v \to w)\) (\(\text{ear}(w \bowtie v)\), respectively) interchangeably.

### 3.1 A high-level description

Since our algorithm is based on Algorithm 3-edge-connectivity of [30], we first briefly review the algorithm.

Starting with the input graph \(G = (V, E)\), the graph is gradually transformed so that vertices that have been confirmed to be belonging to the same 3ecc are merged into one vertex, called a supervertex. Each supervertex is represented by a vertex \(w \in V\) and a set \(\sigma(w)(\subseteq V)\) consisting of vertices that have been confirmed to be belonging to the same 3ecc as \(w\). Initially, each vertex \(w\) is regarded as a supervertex with \(\sigma(w) = \{w\}\). When two supervertices \(w\) and \(u\) are known to be belonging to the same 3ecc, they are merged into a single supervertex with one of them, say \(w\), absorbing the other resulting in \(\sigma(w) := \sigma(w) \cup \sigma(u)\). When that happens, the edges incident on \(u\) become edges incident on \(w\) (the latter are called embodiments.
When a supervertex containing all vertices of a 3ecc is formed, it must be of degree one or two in the transformed graph (corresponding to a bridge or a cut-pair is found). When this condition is detected, the supervertex is separated (ejected) from the graph to become an isolated supervertex. At the end, the graph is transformed into a collection of isolated supervertices each of which contains the vertices of a distinct 3ecc of $G$.

The transformation is carried out by performing a depth-first search over $G$, starting from an arbitrary vertex $r$. At each vertex $w \in V$, when the search backtracks from a child $u$, let $\hat{G}_u$ be the graph to which $G$ has been transformed at that point of time. The subgraph of $G$ induced by the vertex set of $T_u$, $G\langle V_{T_u} \rangle$, has been transformed into a set of isolated supervertices (each of which corresponds to a distinct 3ecc of $G$) and a path of supervertices, $P_u : (u =) u_0 u_1 u_2 \ldots u_k$, called the $u$-path. The $u$-path has the properties summarised in the following lemma.

**Lemma 3.3.** [Lemma 6 of [31]] Let $P_u : (u =) u_0 u_1 u_2 \ldots u_k$ (Figure 1).

(i) $\deg_{\hat{G}_u}(u_0) \geq 1$ and $\deg_{\hat{G}_u}(u_i) \geq 3, 1 \leq i \leq k$;

(ii) for each back-edge $f = (x \bowtie u_i), 0 \leq i \leq k, x \preceq w$ (i.e. $x$ lies on the $r \leadsto_T w$ tree-path);

(iii) $\exists (z \bowtie u_k)$ such that $dfs(z) = \text{lowpt}(u)$.

If $\deg_{\hat{G}_u}(u_0) = 1$, then $(w, u)$ is a bridge, $k = 0$, and $\sigma(u)$ is a 3ecc of $G$. The supervertex $u$ is ejected from the $u$-path to become an isolated supervertex and the $u$-path becomes a null path.

---

1In [30], it is pointed out that Algorithm 3-edge-connectivity can be easily modified to handle non-2-edge-connected graphs.
If $\text{deg}_{G_u}(u_0) = 2$, then $\{(w \to u), (u \to u_1)\}$ or $\{(w \to u), (z \wedge u)\}$, where $z = t(\text{ear}(w \to u))$, is a cut-pair implying $\sigma(u)$ is a 3$\text{ecc}$ of $G$. The supervertex $u$ is ejected from the $u$-path to become an isolated supervertex and the $u$-path is shorten to $u_1u_2\ldots u_k$ in the former case or a null path in the latter case. Next, if $\text{lowpt}(w) \leq \text{lowpt}(u)$, then no edge on the $u$-path can be a cut-edge which implies that the vertices in the supervertices on the $u$-path must all belong to the same 3$\text{ecc}$ as $w$. The supervertices are thus absorbed by $w$. Likewise, if $\text{lowpt}(w) > \text{lowpt}(u)$, then the vertices in the supervertices on the current $w$-path must all belong to the same 3$\text{ecc}$ as $w$ and the supervertices are absorbed by $w$; moreover, $\text{lowpt}(w)$ is updated to $\text{lowpt}(u)$ and the $u$-path becomes the current $w$-path.

When an outgoing back-edge of $w$, $(z \wedge w)$, with $\text{dfs}(z) < \text{lowpt}(w)$ is encountered, vertex $w$ absorbs the current $w$-path because all the supervertices on it belong to the same 3$\text{ecc}$ as $w$; $\text{lowpt}(w)$ and the $w$-path are then updated to $\text{dfs}(z)$ and the null path $w$, respectively.

When an incoming back-edge of $w$, $(w \wedge x)$, is encountered, let $x \in \sigma(w_h)$, where $w_h$ is a supervertex on the current $w$-path $ww_1w_2\ldots w_k (k \geq h)$, then no edge on the path, $ww_1\ldots w_h$, can be a cut-edge. As a result, the vertices in $\sigma(w_i), 1 \leq i \leq h$, must all belong to the same 3$\text{ecc}$ as $w$. The supervertices $w_i, 1 \leq i \leq h$, are thus absorbed by $w$ and the current $w$-path is shortened to $ww_{h+1}w_{h+2}\ldots w_k$.

When the adjacency list of $w$ is completely processed, if $w \neq r$, the depth-first search backtracks to the parent vertex of $w$. Otherwise, the input graph $G$ has been transformed into a collection of isolated supervertices each of which contains the vertices of a distinct 3$\text{ecc}$ of $G$.

Nagamochi et al. [21] pointed out that the graph resulting from $G$ by contracting every 3$\text{ecc}$ into a supervertex is a cactus representation of the cut-pairs in $G$. Hence, the algorithm can be easily extended to generate a cactus representation of $G$ as a byproduct.

It remains to explain how to modify the algorithm so that along with generating the 3$\text{ecc}$s, it also generates a Mader construction sequence for each 3$\text{ecc}$, the bridges and a cactus representation for the cut-pairs by making only one $\text{dfs}$ over the given graph. For clarity, we shall consider these two tasks separately.

### 3.2 Generating construction sequences (positive certificates)

Since each 3$\text{ecc}$ is not a subgraph but a subset of vertices, before discussing generating a Mader construction sequence for a 3$\text{ecc}$, we must address the following question first: “The Mader construction sequence for a 3$\text{ecc}$ is generated based on what set of edges?” Although the edge set of the subgraph of $G$ induced by $\sigma(w)$, i.e. $G_{\langle \sigma(w) \rangle}$, appears to be the correct answer, unfortunately, it is not as $G_{\langle \sigma(w) \rangle}$ may not be
Definition: \( \forall K - u \) and (\( \hat{\text{edges outside}} \) \( G \) in \( G \) path connecting two vertices in \( G \) are only two edge-disjoint paths connecting them in \( G \)\).

Proof: Let \( \text{Lemma 3.4.} \) \( G \) \( w \) be a \( 3\text{-ecc} \) of \( G = (V, E) \) and \( \{(v, w), (\hat{w}, d)\} \) be the cut-pair with \( w, \hat{w} \in \sigma(w) \).

Let \( \hat{G}_{\sigma(w)} = \left\{ \begin{array}{ll}
G_{\sigma(w)} \text{ and a new edge } (w, \hat{w}) \notin E, & \text{if } w \neq \hat{w}; \\
G_{\sigma(w)}, & \text{if } w = \hat{w}.
\end{array} \right. \)

Then, \( \hat{G}_{\sigma(w)} \) is \( 3\text{-edge-connected} \).

Proof: Let \( u, v \in \sigma(w) \). Then, there exist three edge-disjoint \( u - v \) paths in \( G \). If none of the paths uses edges outside \( G_{\sigma(w)} \), then the three paths are edge-disjoint \( u - v \) paths in \( G_{\sigma(w)} \), and hence in \( \hat{G}_{\sigma(w)} \).

Suppose one of the \( u - v \) paths uses edges outside \( G_{\sigma(w)} \). This path must contain the cut-edges \( (v, w) \) and \( (\hat{w}, d) \). Clearly, the edges on the path that are not in \( G_{\sigma(w)} \) all lie on the section connecting \( w \) and \( \hat{w} \).

If \( w = \hat{w} \), discarding the section results in an \( u - v \) path in \( G_{\sigma(w)} \). Hence, there are three edge-disjoint \( u - v \) paths in \( G_{\sigma(w)} \), and hence in \( \hat{G}_{\sigma(w)} \). If \( w \neq \hat{w} \), replacing the section with the new edge \( (w, \hat{w}) \) results in an \( u - v \) path in \( \hat{G}_{\sigma(w)} \). Again, there are three edge-disjoint \( u - v \) paths in \( \hat{G}_{\sigma(w)} \).

Hence, \( \hat{G}_{\sigma(w)} \) is \( 3\text{-edge-connected} \). \( \blacksquare \)

Hence, a Mader construction sequence for \( \sigma(w) \) is constructed based on the edge set of \( \hat{G}_{\sigma(w)} \).

Since a Mader construction sequence starts with a \( K_2^3 \)-subdivision, we consider how to determine the \( K_2^3 \)-subdivision next.

If \( |\sigma(w)| = 1 \), no construction sequence is to be constructed. Let \( |\sigma(w)| > 1 \).

Definition: \( \forall w \in V \), let \( \hat{P}_w = \{P_{\text{ear}(e)} \mid e = (w \rightarrow u) \in E_T \lor e = (u \leftarrow w) \in E \setminus E_T\} \) and \( \hat{P}(w) = \hat{P}_w \).
min \in \hat{\mathcal{P}}_w. When the \textit{dfs} backtracks from vertex \( w \) to its parent vertex or execution of the \textit{dfs} terminates at \( w(=r) \), let \( \hat{\delta}(w) = \{ P \mid P \text{ is an ear } \land t(P) \in \sigma(w) \} \), \( \delta(w) = \hat{\delta}(w) \setminus \{ \hat{\mathcal{P}}(w) \} \) and \( \mathcal{P}(w) = \min \in \delta(w) \).

Intuitively, \( \hat{\mathcal{P}}_w \) is the set of ears that contain a child edge or an outgoing back-edge of \( w \), \( \hat{\delta}(w) \) is the set of ears whose sink is in \( \sigma(w) \) (Note that \( \sigma(w) \) contains only those vertices that are determined to be 3-edge-connected to \( w \) up to the time when the \textit{dfs} backtracks from \( w \)). \( \hat{\mathcal{P}}(w) \) is the lexicographically smallest ear in \( \hat{\mathcal{P}}_w \). \( \mathcal{P}(w) \) is the lexicographically smallest ear in \( \hat{\delta}(w) \) excluding the ear \( \hat{\mathcal{P}}(w) \). If \( w \neq r \) and the parent edge of \( w \) is not a bridge, \( \hat{\delta}(w) = \hat{\delta}(w) \), \( \mathcal{P}(w) \) is the ear containing the parent edge of \( w \), and \( \mathcal{P}(w) \) is the lexicographically smallest ear with the sink in \( \sigma(w) \). Otherwise, \( \delta(w) = \delta(w) \setminus \{ \hat{\mathcal{P}}(w) \} \), \( \hat{\mathcal{P}}(w) \) is the lexicographically smallest ear with the sink and source in \( \sigma(w) \) (it is thus a cycle), and \( \mathcal{P}(w) \) is the lexicographically second smallest ear with the sink in \( \sigma(w) \).

Gabow [8] classified depth-first-search-based graph connectivity algorithms into two types: \textit{lowpoint}-based and path-based. While the algorithm [30] based on which our algorithm is developed is \textit{lowpoint}-based, our certifying algorithm is path-based. This is because it uses ears (paths) to generate a Mader construction sequence. Clearly, \( \text{lowpt}(w) \) can be redefined in terms of ears.

**Lemma 3.5.** Let \( w \in V \) such that \( |\sigma(w)| > 1 \),

(a) \( \text{dfs}(s(\hat{\mathcal{P}}(w))) = \text{lowpt}(w); \) (Figure 1(b))

(b) \( t(\mathcal{P}(w)) \) lies on \( \hat{\mathcal{P}}(w) \).

(c) Let \( w \neq r \) and \( (v \rightarrow w) \) be its parent edge which is not a bridge. Then \( \hat{\mathcal{P}}(w) = P_{\text{ear}(v \rightarrow w)} \).

**Proof:**

(a) By induction on the height of \( w \) in the \textit{dfs} tree.

(b) Suppose \( t(\mathcal{P}(w)) \) does not lie on \( \hat{\mathcal{P}}(w) \). Let \( u = t(\mathcal{P}(w)) \) and \( v \rightarrow u \) be the parent edge of \( u \). Then \( P_{\text{ear}(v \rightarrow u)} \lesssim \mathcal{P}(w) \). Clearly, \( t(P_{\text{ear}(v \rightarrow u)}) \) lies on \( w \Rightarrow u \). As \( u \in \sigma(w) \Rightarrow t(P_{\text{ear}(v \rightarrow u)}) \in \sigma(w) \), \( P_{\text{ear}(v \rightarrow u)} \in \delta(w) \) which contradicts that \( \mathcal{P}(w) \) is the lexicographically smallest ear in \( \delta(w) \).

(c) Immediate from the definitions of \( \hat{\mathcal{P}}(w) \) and \( \text{ear}(v \rightarrow w) \). \( \Box \)

Note that by Lemma [3.5(a)], the \( w \)-path is created from \( \hat{\mathcal{P}}(w) \) through the eject and absorb operation.

The \( K^2 \)-subdivision is constructed as follows:

(i) Let \( \{ e, e' \} \) be the cut-pair separating \( \sigma(w) \) from the rest of the graph: in the course of executing Algorithm 3-edge-connectivity, let \( e \) become \( (v \rightarrow w) \) while \( e' \) become either \( (\tilde{w} \rightarrow d) \) or \( (d \leftarrow \tilde{w}) \) such that \( w, \tilde{w} \in \sigma(w) \) and \( w \preceq \tilde{w} \).
Invariant

“When the dfs backtracks from a vertex \( w(\neq r) \) to its parent vertex, a construction sequence, \( \mathcal{CS}_{\delta(w)} \), has been determined with \( \overrightarrow{P}(w) \) as the first element.”
Note that in this context, \( \sigma(w) \) consists of all the vertices that have been absorbed by \( w \) so far; it becomes a 3\( \text{ecc} \) only after \( w \) is ejected. We call \( \widehat{P}(w) \) the anchor of \( \delta(w) \).

To generate the construction sequences for the 3\( \text{ecc} \)s based on the above discussion, we modify Algorithm 3-edge-connectivity of [30] as follows. Note that we only outline the key idea. The correctness will be proven later on.

(a) \( w \) is a leaf in \( T \): If \( \text{deg}_G(w) \leq 2, \; \sigma(w) = \{w\} \) is a 3\( \text{ecc} \) and no construction sequence is to be constructed. Let \( \text{deg}_G(w) > 2 \). Then \( \delta(w) = \{(u \cap w) \mid P_{\text{ear}(u \cap w)} = (u \cap w)\} \) which is the set of trivial ears whose sink is \( w \). Let \( (u_1 \cap w), (u_2 \cap w), \ldots, (u_{|\delta(w)| - 1} \cap w) \) be the ears in \( \delta(w) \backslash \{\widehat{P}(w)\} \) in an arbitrary order. Then \( CS_{\delta(w)} = \widehat{P}(w) (u_1 \cap w) (u_2 \cap w) \ldots (u_{|\delta(w)| - 1} \cap w) \) and the invariant holds for \( \delta(w) \).

(b) \( w \) is an internal vertex of \( T \): Let \( \{u_i \mid 1 \leq i \leq m\} \) be the set of vertices in \( L[w] \) such that either \( P_{\text{ear}(w \rightarrow u_i)} \in \hat{P}_w \backslash \{\hat{P}(w)\} \) or \( P_{(u_i \cap w)} \in \hat{P}_w \backslash \{\hat{P}(w)\}, 1 \leq i \leq m \).

For \( u_i, 1 \leq i \leq \bar{m}, (i) \) If \( P_{(u_i \cap w)} \in \hat{P}_w \backslash \{\hat{P}(w)\} \), then let \( \delta_{u_i} = \{(u_i \cap w)\} \) and \( CS_{\delta_{u_i}} = (u_i \cap w) \).

(ii) If \( P_{\text{ear}(w \rightarrow u_i)} \in \hat{P}_w \backslash \{\hat{P}(w)\} \), when the \( \text{dfs} \) backtracks from \( u_i \) to \( w \), if \( \text{deg}_{G_{u_i}}(u_i) \leq 2, \sigma(u_i) \) is a 3\( \text{ecc} \). The supervertex \( u_i \) is ejected from \( G_{u_i} \) and \( CS_{\delta(u_i)} \) is used to generate a Mader construction sequence for \( G_{(\sigma(u_i))} \). If \( P_u \) becomes \( \text{null} \), then \( P_{\text{ear}(w \rightarrow u_i)} \) is replaced by a virtual back-edge \( (d \cap w) \), where \( d = t(\text{ear}(w \rightarrow u_i)) \), which becomes Case (i). Hence, let \( \delta_{u_i} = \{(d \cap w)\} \) and \( CS_{\delta_{u_i}} = (d \cap w) \). Otherwise, let the \( u_i \) path be \( P_{u_i} : u_i^1 u_i^2 \ldots u_i^{h_i} \). Then let \( \delta_{u_i} = \{P_{\text{ear}(w \rightarrow u_i)}\} \cup \bigcup_{j=1}^{h_i} \delta(u_i^j) \).

By assumption, the invariant holds for \( u_i^j, 1 \leq j \leq h_i \). Hence, \( CS_{\delta(u_i^j)} \), \( 1 \leq j \leq h_i \), has been constructed. Then \( CS_{\delta_{u_i}} = CS_{\delta(u_i^{h_i})} \cdots CS_{\delta(u_i^2)} \cdots CS_{\delta(u_i)} \).

Without loss of generality, let \( \hat{P}(u_1) = \min_i \{\hat{P}(u_i) \mid 1 \leq i \leq \bar{m}\} \), where \( \hat{P}(u_i) = (u_i \cap w) \), if \( (u_i \cap w) \in E \backslash E_T \), and \( \hat{P}(u_i) = P_{(w \rightarrow u_i)} \), if \( (w \rightarrow u_i) \in E_T \). Since \( \hat{P}(w) < \hat{P}(u_i) \), \( w \) absorbs \( P_{u_i}, 1 \leq i \leq \bar{m} \), resulting in \( \delta(w) = \bigcup_{i=1}^{\bar{m}} \delta_{u_i} = \bigcup_{i=1}^{\bar{m}} (\{\hat{P}(u_i)\} \cup \bigcup_{j=1}^{h_i} \delta(u_i^j)) \) and \( CS_{\delta(w)} = CS_{\delta_{u_1}} \cdots CS_{\delta_{u_h}} \).

If \( w \) has no incoming back-edges with its sink on \( P_w \), then \( \text{dfs} \) backtracks from \( w \) and the invariant holds for \( \delta(w) \). Otherwise, let \( P_w : (w =) w_0 w_1 \ldots w_k \) and \( h \) be the largest index such that there exists \( w \cap w_h \). Let \( \overline{P}(w_h) = \min_{i \in [1, \bar{h}]} \{\overline{P}(w_i) \mid 0 \leq i \leq h\} \) and \( w_i, 1 \leq j \leq \bar{h}, i_j \notin \{h, \ell\} \) be the remaining vertices on the path \( w_0 w_1 \ldots w_h \), where \( \bar{h} = h - 1 \) if \( \ell \neq h \) and \( \bar{h} = h \) if \( \ell = h \). By assumption, the invariant holds for \( w_i, 0 \leq i \leq h \), and hence \( CS_{\delta(w_i)} \), \( 0 \leq i \leq h \), are constructed.
After vertex \( w \) absorbed \( w_i, 1 \leq i \leq h \), \( CS_{\delta(w)} = CS_{\delta(w_1)}CS_{\delta(w_h)}CS_{\delta(w_{i+1})} \ldots CS_{\delta(w_{h-1})} \) if \( \ell \neq h \), and \( CS_{\delta(w)} = CS_{\delta(w_h)}CS_{\delta(w_{i+1})} \ldots CS_{\delta(w_{h-1})} \) if \( \ell = h \), and the invariant holds for \( \delta(w) \).

When the \( dfs \) terminates at \( r \), the \( 3eccs \) of \( G \) and their Mader construction sequences are generated.

### 3.2.1 The certifying algorithm

The following is a pseudo-code of the algorithm which is based on the pseudo-code in [30]. The new instructions are marked with \( \bullet \). As with [30], for clarity, the algorithm is presented without taking parallel edges into consideration. With a simple modification, the algorithm can handle parallel edges.

In Procedure 3-edge-connect-\( CS(w,v) \), the for loop processes the adjacency list \( L[w] \) of vertex \( w \). The if-part of the first if statement in the for loop deals with unvisited vertices leading to non-trivial ears \( P \) with \( t(P) = w \) or \( P = \hat{P}(w) \) while the else-part deals with visited vertices leading to trivial ears \( P \) with \( t(P) = w \) or \( P = \hat{P}(w) \). Procedure Absorb-ear absorbs the entire \( u \)-path or \( w \)-path and merges the \( CSs \) of the supervertices involved to update \( CS_{\delta(w)} \). Procedure Absorb-path absorbs the section \((w =)w_0w_1w_2 \ldots w_h \) of the \( w \)-path and merges \( CS_{\delta(w_i)}, 0 \leq i \leq h \), to update \( CS_{\delta(w)} \). Procedure Gen-\( CS \) converts \( CS_{\delta(u)} \) of a 3ecc \( \sigma(u) \) into a Mader construction sequence of \( G_\sigma(u) \) by adding paths or ears to create the \( K_2^3 \)-subdivision that leads the construction sequence.

**Algorithm 3-certifying-edge-connectivity**

**Input:** A connected graph \( G = (V,E) \) represented by adjacency lists \( L[w], \forall w \in V \)

**Output:** \( \{CS_G \mid G' \text{ is a 2ecc of } G\} \) a cactus representation of the cut-pairs for each 2ecc \( G' \) of \( G \), and the bridges in \( G \)

begin
  for every \( u \in V \) do dfs\( (u) := 0 \); parent\( (u) := \_ \); lowpt\( (u) := \infty \); // initialization; \_ = undefined
  \( \bullet \)
  \( P(u) := \hat{P}(u) := \_ \); \( CS_{\delta(u)} := \_ \); // note: \( \_ \preceq P ; \forall f \in E \setminus E_T ; \_ \preceq CS_{\delta(u)} = CS_{\delta(u)} \preceq CS_{\delta(u)} \)
  \( \sigma(u) := \{u\} \); Inc\( u := 0 \); \( P_u := u \);
  cnt := 1; // dfs number counter // Bridges := \_ // to store the bridges in \( G \)

  3-edge-connect-\( CS(r, L_u) \);
  \( CS_{\delta(r)} := P(r)CS_{\delta(r)} \); // Finalize \( CS_{\delta(r)} \); // theorem

end.

**Procedure 3-edge-connect-\( CS(w,v) \)**

begin
  dfs\( (w) := cnt \); cnt := cnt + 1; parent\( (w) := v \); lowpt\( (w) := dfs\( (w) \); // initialization
  \( \bullet \) if \( (w \neq r) \) then ear\( (v \rightarrow w) := \_ \);
  for each \( u \in L_u \) do // pick the next vertex \( u \) in the adjacency list of \( w \)
    if \( (dfs(u) = 0) \) then // \( u \) is unvisited
      3-edge-connect-\( CS(u,v) \);
      if \( (s(\hat{P}(u)) = u \cup \hat{P}(u) := \_ \) then
        Gen-\( CS(u,v,P_u) \);
      \( \bullet \) if \( (P(u) \preceq P(v)) \) then
        Absorb-ear\( (w, \hat{P}(u), w + P_u) \)
      else // \( (P(u) \neq P(w)) \);
        Absorb-ear\( (w, \hat{P}(w), P_u) \);

end.
\[ P_w := w + P_u; \hat{P}(w) := \hat{P}(u); \]

\[ \text{if} (w \neq r) \text{ then } \text{ear}(v \rightarrow w) := \text{ear}(w \rightarrow u); \]

\[ \text{else if} (df(s(u) < df(s(w)) \wedge u \neq \text{parent}(w)) \text{ then} \]

\[ (u \wedge w) \leq P(w) \]

\[ \text{Absorb-ear}(w, \hat{P}(w), P_w); \]

\[ P_w := w; \hat{P}(w) := u \wedge w; \]

\[ \text{else} \]

\[ \text{Absorb-ear}(w, u \wedge w, w); \]

\[ \text{end} \]

\[ \text{else Inc}_w := \text{Inc}_w \cup \{(w \wedge u)\}; \]

if \((P_w \neq w) \wedge (\text{Inc}_w \neq \emptyset)\) then \text{Absorb-path}(w, P_w, \text{Inc}_w); // dealing with incoming back-edges

end. */ of Procedure 3-edge-connect-CS */

Procedure Gen-CS\((w, u, P_u)\)
begin \(// P_u := (u = u_0 u_1 \ldots u_h)\); eject super-vertex \(u\) from \(P_u\); finalize \(\text{CS}_{\sigma(u)}\)

\[ \text{output}(\sigma(u)); \]

\[ \text{if} \quad ((s(\hat{P}(u)) = u \vee (\hat{P}(u) = \bot)) \text{ then} \]

\[ \text{CS}_{\delta(u)} := \hat{P}(u) \wedge \text{CS}_{\delta(u)}; \]

\[ P(u) := \text{ear}(w \rightarrow u) := \bot; \quad P_u := \text{nil}; \]

\[ \text{Bridges} := \text{Bridges} \cup \{(w, u)\}; \]

\[ \quad (w, u) \text{ is a bridge} \]

\[ \text{else} \]

\[ \quad (\text{deg}_{\hat{P}}(u) = 2)\]

\[ \text{if} \quad (P_u = u) \text{ then} \]

\[ \quad \text{cut-pair} := \{(w \rightarrow u), (d \wedge \hat{u})\}, \text{ where} (d \wedge \hat{u}) := \text{ear}(w \rightarrow u) ; \]

\[ \quad \hat{u} := s(\text{ear}(w \rightarrow u)); \ d := t(\text{ear}(w \rightarrow u)); \]

\[ \quad \hat{P}(u) := (d \wedge w); \quad P_u := \text{nil}; \]

\[ \quad \text{else} \]

\[ \quad \text{parent}(u_1) := w; \]

\[ \quad \text{ear}(w \rightarrow u_1) := \text{ear}(w \rightarrow u); \]

\[ \quad \text{replace} \quad (d \rightarrow P_{\text{ear}(w \rightarrow u)} w) \text{ with virtual edge} (d \wedge w) \]

\[ \quad \text{remove} \quad u \text{ from} \ P_u \]

\[ \text{if} \quad (u \neq \hat{u}) \text{ then} \]

\[ \text{if} \quad (t(\hat{P}(u)) \neq u) \text{ then} \]

\[ \text{CS}_{\delta(u)} := (u \rightarrow \hat{u} \hat{u}) (u, \hat{u}) \text{ CS}_{\delta(u)}; \]

\[ \text{else} \]

\[ \text{CS}_{\delta(u)} := \text{CS}_{\delta(u)} \text{ such that} \text{CS}_{\delta(u)} = \text{CS}_{\delta(u)} \text{ CS}_{\delta(u)}; \]

\[ \text{CS}_{\delta(u)} := (u \rightarrow \hat{u} \hat{u}) (u, \hat{u}) \text{ CS}_{\delta(u)}; \]

\[ \text{end} \]

end.

Procedure Absorb-ear\((w, \hat{P}; P)\)
\(//\) absorb the entire \(P\) which is \(P_w\) or \(w + P_u\) with \(P\) being \(\hat{P}(w)\) or \(\hat{P}(u)\), respectively
begin \(// P := (x_0 x_1 x_2 \ldots x_{k-1} x_k)\) such that \(x_i.\text{next} = x_{i+1}, 0 \leq i < k\) and \(x_k.\text{next} = \bot\)

\[ \text{CS} := \bot; \]

\[ x := \text{head}(P); \]

while \((x.\text{next} \neq \bot)\) do

\[ x := x.\text{next}; \]

\[ \sigma(w) := \sigma(w) \cup \sigma(x); \]

\[ \text{CS} := \text{CS}_{\sigma(zx)} \text{ CS}_z; \]

\[ \text{CS} := \hat{P} \text{ CS}_z; \]

\[ \text{if} \quad (\hat{P} < \hat{P}(w)) \text{ then} \]

\[ \text{CS}_{\delta(w)} := \text{CS}_{\delta(w)} \hat{P}(w) := \hat{P}; \]

\[ \text{else} \]

\[ \text{CS}_{\delta(w)} := \text{CS}_{\delta(w)} \text{ CS}_z; \]

\[ \text{end} \]

end.

Procedure Absorb-path\((w, P_w, \text{Inc})\)
\(//\) absorb a section of the \(w\)-path \(P_w\)
begin \(// P_w := (w = w_0 w_1 \ldots w_k)\) such that \(w_i.\text{next} = w_{i+1}, 0 \leq i < k\) and \(w_k.\text{next} = \bot\)

\[ h := 0; \ w := w_0; \]

\[ \text{for each} \quad ((w \wedge x) \in \text{Inc}) \text{ do} \]

\[ \text{while} \quad ((w.\text{next} \neq \bot) \land (w.\text{next} \leq x)) \quad \text{do} \quad (h := h + 1; \ w := w.\text{next}); \]

\[ \hat{P}(w_j) := \text{min} \{ \hat{P}(w_j) | 0 \leq j \leq h \}; \]

\[ \text{if} \quad (\hat{P}(w_j) \neq \hat{P}(w_j)) \text{ then} \]

\[ \text{CS}_{\delta(w)} := \text{CS}_{\delta(w)} \text{ CS}_{\delta(w)} \text{ else} \text{CS}_{\delta(w)} := \text{CS}_{\delta(w)}; \]

\[ \text{for} \quad j := 0 \text{ to} h - 1 \text{ do} \]

\[ \text{if} \quad (\hat{P}(w_j) \neq \hat{P}(w_j)) \text{ then} \text{CS}_{\delta(w)} := \text{CS}_{\delta(w)} \text{ CS}_{\delta(w)}; \]

\[ \text{end} \]

end.
Remark: When the condition \(\text{deg}_{\hat{G}_u}(u) = 2\) is detected, as a result of ejecting \(\sigma(u)\), the two corresponding cut-edges are removed and a new (virtual) edge is created in \(\hat{G}_u \setminus \{u\}\) to connect the two end-vertices of the cut-pairs that are not in \(\sigma(u)\). Fortunately, we do not need to update the adjacency lists involved to accommodate the changes. We just update \(\hat{P}(u)\) as it contains the cut-pairs and their end-vertices. This is done in Procedure \text{gen-CS}\) as follows:

- \(\{(w \to u), (d \swarrow u)\}\) is the cut-pair in \(\hat{G}_u\): Then \(\{(w \to u), (d \swarrow \hat{u})\}\) is the cut-pair in \(G\) and the new edge to be added is \((d \swarrow w)\). Since \(\hat{P}(u)\) consists of \((w \to u), u \sim_T \hat{u}\) and \((d \swarrow \hat{u})\) which are replaced by \((d \swarrow w)\), we thus update \(\hat{P}(u)\) with \((d \swarrow w)\).

Let \((d \swarrow w_x)\) be the embodiment of \((d \swarrow w)\) when the incoming back-edges of \(d\) are examined at vertex \(d\). Since \(L[d]\) was not updated to replace node \(\hat{u}\) with node \(w\) when vertex \(w\) was the current vertex of the \(dfs\), it is \((d \swarrow \hat{u})\) instead of \((d \swarrow w)\) that will be encountered when \(L[d]\) is processed. However, as \((d \swarrow w)\) is an embodiment of \((d \swarrow \hat{u})\), \((d \swarrow w_x)\) is also an embodiment of \((d \swarrow \hat{u})\).

Hence, \(w_x\) can be correctly identified using \((d \swarrow \hat{u})\).

- \(\{(w \to u), (u \to u_1)\}\) is the cut-pair in \(\hat{G}_u\): Then \(\{(w \to u), (\hat{u} \to u_1)\}\) is the cut-pair in \(G\) and the section \((w \to u)u \sim_T \hat{u}(\hat{u} \to u_1)\) on \(\hat{P}(u)(= \hat{P}(u_1))\) is replaced by \((w \to u_1)\). We thus let \(\text{parent}(u_1) = w\) and \(\text{ear}(w \to u_1) = \text{ear}(w \to u)\).

We shall prove that the invariant holds for \(\delta(w)\), for every vertex \(w\).

If \(\text{deg}_G(w) \leq 2\), \(\delta(w) = \emptyset\) and no \(CS_{\delta(w)}\) is needed. Let \(\text{deg}_G(w) > 2\). First, consider the leaves of \(T\).

**Lemma 3.6.** Let \(w\) be a leaf in \(T\) such that \(\text{deg}_G(w) > 2\). Let \(P_1, P_2, \ldots, P_{\text{deg}_G(w)-3}\) be the ears in \(\delta(w) \setminus \{\hat{P}(w)\}\) in arbitrary order. Then \(CS_{\delta(w)} = \{\hat{P}(w)\}P_1P_2 \ldots P_{\text{deg}_G(w)-3}\). (Figure 4(a)).

**Proof:** Since \(w\) is a leaf, \(\sigma(w) = \{w\}\). Therefore, \(\delta(w) = \{P \mid P\text{ is an ear and } t(P) = w\} = \{f \mid f = (u \swarrow w) \in E \setminus E_T \} \setminus \{\hat{f}\}\), where \(P_{\hat{f}} = \hat{P}(w)\). Clearly, after \(\hat{P}(w)\) is added to the construction sequence, \(w\) becomes a branch vertex. The remaining ears (back-edges) \(P_1, P_2, \ldots, P_{\text{deg}_G(w)-3}\) in \(\delta(w) \setminus \{\hat{P}(w)\}\) can be added to the construction sequence as Mader paths by operation (i) or (ii). The lemma follows. 

Next, consider the internal vertices. Let \(w\) be an internal vertex of \(T\) with \(\text{deg}_G(w) > 2\). In processing \(L[w]\), vertices that are children of \(w\) or form an outgoing back-edge of \(w\) with \(w\) are processed first. The vertices that form an incoming back-edge of \(w\) with \(w\) are considered later.

**Lemma 3.7.** Let \(w\) be an internal vertex of \(T\) such that \(\text{deg}_G(w) > 2\).

Let \(u \in L[w]\).
Figure 4: \( \overrightarrow{P}(w) \), where (a) : \( w \) is a leaf; (b), (c) : \( w \) is an internal vertex.

(i) If \( u \) is a child of \( w \) and \( u_1, u_2, \ldots, u_h \) is the \( u \)-path after the dfs backtracked from \( u \) to \( w \) and \( u \) is ejected if \( \deg_{G_u}(u) = 2 \). Suppose the invariant holds for each \( u_j, 1 \leq j \leq h \).

Let \( \delta_u = \{ \text{ear}(w \rightarrow u) \} \cup \bigcup_{j=1}^{h} \delta(u_j) \). Then \( CS_{\delta_u} = P_{\text{ear}(w \rightarrow u)} \cup CS_{\delta(u_h)}CS_{\delta(u_{h-1})} \cdots CS_{\delta(u_1)} \).

(ii) If \((u \sim w) \in E \setminus E_T\), let \( \delta_u = \{(u \sim w)\} \). Then \( CS_{\delta_u} = (u \sim w) \).

**Proof:**

(i) After \( P_{\text{ear}(w \rightarrow u)} \) is added to the construction sequence, \( w \) becomes a branch vertex.

For each \( \overrightarrow{P}(u_j), 1 \leq j \leq h \), since \( t(\overrightarrow{P}(u_j)) \subseteq \sigma(u_j) \) and by Lemma 3.3(ii), \( s(P_{\text{ear}(w \rightarrow u)}) \leq s(\overrightarrow{P}(u_j)) \leq w < t(\overrightarrow{P}(u_j)) \), \( \overrightarrow{P}(u_j) \) can be added to the construction sequence by operation (ii) or (iii).

As the invariant holds for \( u_j \), the remaining ears in \( \delta(u_j) \) can be added to the construction sequence after \( \overrightarrow{P}(u_j) \). Hence, \( CS_{\delta_u} = P_{\text{ear}(w \rightarrow u)} \cup CS_{\delta(u_h)}CS_{\delta(u_{h-1})} \cdots CS_{\delta(u_1)} \).

(ii) Obvious. ■

Now, consider the incoming back-edges of \( w \).

**Lemma 3.8.** Let \( w \) be an internal vertex of \( T \). Let \( \hat{G} \) be the graph to which \( G \) has been transformed after the child edges and outgoing back-edges of \( w \) are processed and \( w(=w_0), w_1, w_2, \ldots, w_k(k \geq 1) \) be the \( w \)-path after the child \( u \) of \( w \) on the path is ejected if \( \deg_{G_u}(u) = 2 \).

Suppose the invariant holds for \( w_i, 0 \leq i \leq k \).

Let \( w \sim w_h \) be the incoming back-edge of \( w \) in \( \hat{G} \) with the largest index \( h \).

Let \( \overrightarrow{P}(w_i) = \min \{ \overrightarrow{P}(w_i) \mid 0 \leq i \leq h \} \), and let \( \overrightarrow{P}(w_{i_1}) \overrightarrow{P}(w_{i_2}) \ldots \overrightarrow{P}(w_{i_h}) \) be the ears in \( \{ \overrightarrow{P}(w_i) \mid 0 \leq i \leq h \} \setminus \{ \overrightarrow{P}(w_{i_1}), \overrightarrow{P}(w_{i_2}), \ldots, \overrightarrow{P}(w_{i_h}) \} \) in arbitrary order, where \( h = k \) or \( h - 1 \) depending on whether \( w_{i_e} = w_h \).
Then, after vertex \( w \) absorbed \( w_i, 1 \leq i \leq h \), \( \bar{P}(w) = \bar{P}(w_h) \) and

\[
CS_{\delta(w)} = \begin{cases} 
CS_{\delta(w_1)}CS_{\delta(w_2)}CS_{\delta(w_3)}\ldots CS_{\delta(w_k)}, & \text{if } \ell \neq h; \\
CS_{\delta(w_h)}CS_{\delta(w_1)}CS_{\delta(w_2)}\ldots CS_{\delta(w_h)}, & \text{if } \ell = h.
\end{cases}
\]

(Figure 4(c)).

**Proof:** By assumption, the invariant holds for \( w_i, 0 \leq i \leq k \). Hence, the invariant holds for \( w_h \). It follows that after \( \bar{P}(w_h) \) is added to the construction sequence, the remaining ears in \( \delta(w_h) \) can be added to the sequence as Mader paths. This produces the construction sequence \( CS_{\delta(w_h)} \).

If \( \bar{P}(w_h) \neq \bar{P}(w) \), then \( s(\bar{P}(w_h)) \leq s(\bar{P}(w)) \leq t(\bar{P}(w_h)) \). Since the invariant holds for \( w_h \), the remaining ears in \( \delta(w_h) \), can be added to the sequence as Mader paths. This produces the construction sequence \( CS_{\delta(w_h)} \).

After that, \( w \) becomes a branch vertex. The ears \( \bar{P}(w_i), 1 \leq j \leq h \), can be added to the sequence as Mader path by operation (ii) or (iii) after \( \bar{P}(w_i) \). Since the invariant holds for \( w_i, 1 \leq j \leq h \), after ear \( \bar{P}(w_i) \) is added, the remaining ears in \( \delta(w_i) \) can be added to the construction sequence as Mader paths. We thus have:

\[
\begin{align*}
CS_{\delta(w)} &= \begin{cases} 
CS_{\delta(w_1)}CS_{\delta(w_2)}CS_{\delta(w_3)}\ldots CS_{\delta(w_k)}, & \text{if } \ell \neq h; \\
CS_{\delta(w_h)}CS_{\delta(w_1)}CS_{\delta(w_2)}\ldots CS_{\delta(w_h)}, & \text{if } \ell = h.
\end{cases}
\end{align*}
\]

is a construction sequence for \( \bigcup_{i=0}^{h} \delta(w_i) \).

After vertex \( w \) absorbed \( w_i, 1 \leq i \leq h \), \( \delta(w) = \bigcup_{i=0}^{h} \delta(w_i) \) which implies that \( \delta(w) = \bigcup_{i=0}^{h} \delta(w_i) \setminus \{ \bar{P}(w) \} = \bigcup_{i=0}^{h} (\delta(w_i) \setminus \{ \bar{P}(w_i) \}) \). Since \( w_i, 1 \leq i \leq h \), lies on the \( w \)-path, \( \bar{P}(w_i) = \bar{P}(w_0) = \bar{P}(w) \). It follows that \( \delta(w) = \bigcup_{i=0}^{h} (\delta(w_i) \setminus \{ \bar{P}(w_i) \}) = \bigcup_{i=0}^{h} \delta(w_i) \). The above construction sequence is thus \( CS_{\delta(w)} \).

Since \( \bar{P}(w) = \min_{\prec} \{ \bar{P}(w_i) \mid 0 \leq i \leq h \} \) and \( \bar{P}(w) = \min_{\prec} \delta(w_i) \), by the transitivity of \( \prec \), \( \bar{P}(w_i) = \min_{\prec} \bigcup_{i=0}^{h} \delta(w_i) = \min_{\prec} \delta(w) \) which implies that \( \bar{P}(w) = \bar{P}(w_h) \). The lemma thus follows.

**Lemma 3.9.** Let \( w \in V \). When the dfs backtracks from \( w \) to its parent \( v \) or terminates execution if \( w = r \), suppose the invariant holds for \( w \).

(i) if \( \deg_{G_w}(w) = 1 \) or \( w = r \), then \( \bar{P}(w)CS_{\delta(w)} \) is a construction sequence of \( \bar{G}_{\sigma(w)} \); (Figure 3(e), (f))

(ii) if \( \deg_{G_w}(w) = 2 \), let \( P_w : (w =) w_0w_1 \ldots w_k \) be the \( w \)-path and \( \bar{w} = \text{parent}(w_1) \) or \( \bar{w} = t(\text{ear}(v \rightarrow w)) \).

(a) If \( w = \bar{w} \), then \( CS_{\delta(w)} \) is a construction sequence of \( \bar{G}_{\sigma(w)} \). (Figure 3(c), (d))

(b) If \( w \neq \bar{w} \), then (Figure 3(a), (b))
Proof:

(i) If \( \deg_{G_w}(w) = 1 \), the parent-edge of \( w \) is a bridge which implies that \( \lowpt(w) = \dfs(w) \). By Lemma 3.5(a), \( s(\hat{P}(w)) = w \) which implies that \( \hat{P}(w) \) is a cycle with \( t(\hat{P}(w)) = w \). Hence, \( \hat{P}(w) \in \delta(w) \). By Lemma 3.5(b), \( t(\hat{P}(w)) \) lies on \( \hat{P}(w) \). Clearly, \( s(\hat{P}(w)) = w \) and \( \hat{P}(w) \) is not a cycle. Hence, \( t(\hat{P}(w)) \neq w \) and \( \hat{P}(w) \) and \( \hat{P}(w) \) form a \( K_2^3 \)-subdivision (Figure 3(c)). By assumption, \( CS_{w}(w) \) is a construction sequence for \( \delta(w) \). As \( \hat{P}(w) \) is the first ear in \( CS_{w}(w) \), \( \hat{P}(w) \) is a construction sequence for \( \delta(w) \) starting with the aforementioned \( K_2^3 \)-subdivision. Since after \( w \) is ejected, \( \sigma(w) \) is a \( 3\text{ecc} \), \( \hat{P}(w) \) \( CS_{w}(w) \) is thus a construction sequence for \( \hat{G}(\sigma(w)) \).

The proof for the case where \( w = r \) is similar (Figure 3(f)).

(ii) if \( \deg_{G_w}(w) = 2 \), then \( \pea(w) = \hat{P}(w) \) which implies that \( t(\hat{P}(w)) \prec w \). Hence, \( \hat{P}(w) \notin \delta(w) \) which implies that \( \delta(w) = \delta(w) \).

(a) \( w = \hat{w} \): Then \( w \) has no incoming back-edges originated from the \( w \)-path (Figure 3(c),(d)) which implies that \( w \) is the only vertex on \( \hat{P}(w) \) that is in \( \sigma(w) \). Since \( t(\hat{P}(w)) \) lies on \( \hat{P}(w) \) and \( \hat{P}(w) \in \delta(w), \) therefore \( t(\hat{P}(w)) = w \) which implies that \( \hat{P}(w) = \pea(w) \) for some child \( u \) of \( w \). By Lemma 3.7 \( CS_{w} = \pea(w) CS_{\delta(w)} \) \( CS_{\delta(w)} \) \( CS_{\delta(w)} \) \( CS_{\delta(w)} \) thus has \( \hat{P}(w) \) as the two leading ears. Since the parent-edge of \( w \) is a cut-edge, \( s(\hat{P}(w)) = w \) which implies that \( \hat{P}(w) \) is a cycle. By Lemma 3.5(b), \( t(\hat{P}(w)) \) lies on \( \hat{P}(w) \). Since \( t(\hat{P}(w)) \in \sigma(u), \) \( t(\hat{P}(w)) \neq w \). By Lemma 3.5(ii), \( s(\hat{P}(u)) = w \). Hence, \( \hat{P}(w) \) and \( \hat{P}(w) \) form a \( K_2^3 \)-subdivision and \( CS_{w}(w) \) starts with the \( K_2^3 \)-subdivision. After \( w \) is ejected, \( \sigma(w) \) is a \( 3\text{ecc} \), \( CS_{w}(w) \) is thus a construction sequence of \( \hat{G}(\sigma(w)) \).
(b) \( w \neq \hat{w} \): Then \( w \) must have absorbed a section of \( \mathcal{P}_w \). Let the section be \( (w) = w_0w_1 \ldots w_h \). By assumption, the invariant holds for \( w_i, 0 \leq i \leq h \). Hence, by Lemma 3.8,

\[
\mathcal{CS}_{\delta(w)} = \begin{cases} 
\mathcal{CS}_{\delta(w_0)}\mathcal{CS}_{\delta(w_1)}\mathcal{CS}_{\delta(w_2)} \ldots \mathcal{CS}_{\delta(w_h)}, & \text{if } \ell \neq h; \\
\mathcal{CS}_{\delta(w_h)}\mathcal{CS}_{\delta(w_{i_1})}\mathcal{CS}_{\delta(w_{i_2})} \ldots \mathcal{CS}_{\delta(w_{i_k})}, & \text{if } \ell = h 
\end{cases}
\]

\( \{w_0, w_1, \ldots, w_h\} \setminus \{w_\ell, w_h\} \), is a construction sequence for \( \delta(w) \) and hence for \( \hat{\delta}(w) \) as \( \hat{\delta}(w) = \delta(w) \). If \( w_\ell \neq w \), then \( w \prec w_\ell \). By Lemma 3.3 (b), \( t(\hat{\mathcal{P}}(w_\ell)) \) lies on \( \hat{\mathcal{P}}(w_\ell) \) which is \( \hat{\mathcal{P}}(w) \).

As \( t(\hat{\mathcal{P}}(w_\ell)) \in \sigma(w), w \prec t(\hat{\mathcal{P}}(w_\ell)) \leq \hat{w} \) which implies that \( t(\hat{\mathcal{P}}(w_\ell)) \) lies on \( w \sim_T \hat{w} \).

Moreover, as \( \langle v \to w \rangle \) is a cut-edge, \( s(\hat{\mathcal{P}}(w_\ell)) = w \). Hence, the path \( w \sim_T \hat{w}, \hat{\mathcal{P}}(w_\ell) \), and the virtual edge \( (w, \hat{w}) \) form a \( K_2^3 \)-subdivision. Note that as \( \hat{\mathcal{P}}(w) \neq \hat{\delta}(w) \) and the path \( w \sim_T \hat{w} \) lies on \( \hat{\mathcal{P}}(w) \), the \( K_2^3 \)-subdivision and the ears in \( \hat{\delta}(w) \), excluding \( \hat{\mathcal{P}}(w) \), are disjoint.

Since \( \hat{\mathcal{P}}(w_\ell)(= \hat{\mathcal{P}}(w)) \) is the anchor of \( \mathcal{CS}_{\delta(w)} \), after \( w \) is ejected, \( \sigma(w) \) becomes a 3\( \text{ecc} \) and \( \langle w \sim_T \hat{w} \rangle(w, \hat{w}) \mathcal{CS}_{\delta(w)} \) is a construction sequence of \( \hat{G}(\sigma(w)) \).

If \( w_\ell = w \), then \( s(\hat{\mathcal{P}}(w_\ell)) = t(\hat{\mathcal{P}}(w_\ell)) = w. \hat{\mathcal{P}}(w_\ell) \) is thus a cycle. By Lemma 3.8

\[
\mathcal{CS}_{\delta(w)} = \mathcal{CS}_{\delta(w)} \mathcal{CS}_{\delta(w_1)} \mathcal{CS}_{\delta(w_2)} \ldots \mathcal{CS}_{\delta(w_h)} \text{ as } \ell \neq h. \]

Unfortunately, \( \langle w \sim_T \hat{w} \rangle(w, \hat{w}) \mathcal{CS}_{\delta(w)} \) is not a construction sequence of \( \hat{G}(\sigma(w)) \) as \( \hat{\mathcal{P}}(w_\ell) \) (the anchor of \( \mathcal{CS}_{\delta(w)} \)), \( \langle w \sim_T \hat{w} \rangle \) and \( \langle w, \hat{w} \rangle \) do not form a \( K_2^3 \)-subdivision. Consider the sequence \( \mathcal{CS}'_{\delta(w)} = \mathcal{CS}_{\delta(w_0)} \mathcal{CS}_{\delta(w_1)} \mathcal{CS}_{\delta(w_2)} \ldots \mathcal{CS}_{\delta(w_h)} \mathcal{CS}_{\delta(w)} \). Clearly, \( \mathcal{CS}'_{\delta(w)} \) is also a construction sequence of \( \delta(w) \) as after adding \( \mathcal{CS}_{\delta(w)} \mathcal{CS}_{\delta(w_1)} \mathcal{CS}_{\delta(w_2)} \ldots \mathcal{CS}_{\delta(w_h)} \), \( w \) becomes a branch vertex; the anchor \( \hat{\mathcal{P}}(w_\ell) \) of \( \mathcal{CS}_{\delta(w)} \) can thus be added to the construction sequence by operation (i) following by the remaining ears in \( \delta(w_\ell) \). Since \( h \neq \ell \), by an argument similar to the above case, it is easily verified that \( \hat{\mathcal{P}}(w_h)(= \hat{\mathcal{P}}(w)) \) (the anchor of \( \mathcal{CS}'_{\delta(w)} \)), \( w \sim_T \hat{w} \), \( w \hat{\mathcal{P}}(w_\ell) \), and \( \hat{w} \) form a \( K_2^3 \)-subdivision. Hence, \( \langle w \sim_T \hat{w} \rangle(w, \hat{w}) \mathcal{CS}'_{\delta(w)} \) is a construction sequence of \( \hat{G}(\sigma(w)) \).

The construction sequence \( \mathcal{CS}_{\delta(w)} \) can be conveniently represented by a linked list. The ordering of the ears in the list obeys that of the ears in \( \mathcal{CS}_{\delta(w)} \). Hence, \( \hat{\mathcal{P}}(w) \) is the \textit{first node} in the list. An ear \( P_f \) in the list can be conveniently represented by \( f \) as \( P_f \) can be easily determined through \( f \) and the arrays parent and ear. The node for ear \( P_f \) is created when \( f \) is encountered. To efficiently handle the situation described in Case (ii)(b) of Lemma 3.9, where \( \mathcal{CS}_{\delta(w)} \) is to be replaced by \( \mathcal{CS}'_{\delta(w)} \), a pointer is kept in the first node of \( \mathcal{CS}_{\delta(w)} \) pointing at the first node of \( \mathcal{CS}_{\delta(w_h)} \). The pointer is created when \( \mathcal{CS}_{\delta(w_h)} \) is appended to \( \mathcal{CS}_{\delta(w)} \).

**Theorem 3.10.** Algorithm Certifying-3-edge-connectivity constructs a Mader construction sequence for every 3-edge-connected component of \( G \).
Proof: (By induction on the height of $T$ to prove that the invariant holds for all vertices)

At a leaf $w$, the else-part of the if statement in the for loop of Procedure 3-edge-connect-$CS(w,v)$ is executed for each $u \in L[w]$. It is easily verified that when $L[w]$ is completely processed, $\hat{P}(w)$, $\overrightarrow{P}(w)$ are correctly computed and $CS_{\delta(w)}$ satisfies Lemma 3.6. Hence, the invariant holds for $w$.

Let $w$ be an internal vertex such that $deg_G(w) > 2$.

Let $u_1, u_2, \ldots, u_m$ be the vertices in $L[w]$ listed in the order they are visited by the $dfs$ such that either $(w \rightarrow u_i) \in E_T$ or $(u_i \rightarrow w) \in E \setminus E_T$, $1 \leq j \leq m$.

Let $P_i(w) = P_{ear(w \rightarrow u_i)}$ or $P_i(w) = (u_i \rightarrow w)$ whichever exists, and

(i) if $P_i(w) = P_{ear(w \rightarrow u_i)}$, let the $u_i$-path be $P_{u_i} : u_1^i u_2^i \ldots u_{h_i}^i$ after $u_i$ is ejected if $deg_{G_{u_i}}(u_i) \leq 2$ and $\delta_{u_i} = \{P_{ear(w \rightarrow u_i)}\} \cup \bigcup_{j=1}^{h_i} \delta(u_i^j)$; (ii) if $P_i(w) = (u_i \rightarrow w)$, let $\delta_{u_i} = \{(u_i \rightarrow w)\}$.

Suppose the invariant holds for $u_j^i, 1 \leq j \leq h_i, 1 \leq i \leq m$.

We shall apply induction on $q$ to prove the following assertion:

After processing the vertices $u_1, u_2, \ldots, u_q$ ($q \leq m$),

$\hat{P}(w) = P_{\hat{\ell}(w)} = \min_{\ell} \{P_i(w) | 1 \leq i \leq q\}$,

$\overrightarrow{P}(w) = P_{\hat{\ell}(w)} = \min_{\ell} \{\{P_i(w) | 1 \leq i \leq q \land i \neq \hat{\ell}\}\}$,

$\delta(w) = \bigcup\{\delta_{u_i} | 1 \leq i \leq q \land i \neq \hat{\ell}\}$, and

$CS_{\delta(w)} = CS_{\delta_{u_{1}}} CS_{\delta_{u_{1,1}}} \ldots CS_{\delta_{u_{q-2}}}$, where $\{u_{i_1}, u_{i_2}, \ldots, u_{i_{q-2}}\} = \{u_1, u_2, \ldots, u_q\} \setminus \{u_{\ell}, u_{\hat{\ell}}\}$ such that $CS_{\delta_{u_{\ell}}} = \overrightarrow{P}(w)CS_{\delta(u_{\ell}^{h_{\ell}})} CS_{\delta(u_{\ell}^{h_{\ell}-1})} \ldots CS_{\delta(u_{\ell}^1)}$ and

$CS_{\delta_{u_{i_j}}} = P_{i_j}(w)CS_{\delta(u_{i_j}^{h_{i_j}})} CS_{\delta(u_{i_j}^{h_{i_j}-1})} \ldots CS_{\delta(u_{i_j}^1)}$, $1 \leq j \leq q - 2$, where $h_{i_j} = h_j$ if $u_{i_j} = u_{i_j}$.

When $q = 1$,

(i) if $(w \rightarrow u_1) \in E_T$, the then-part of the if statement in the for loop in Procedure 3-edge-connect-$CS$ is executed. When the $dfs$ backtracks from $u_1$ to $w$, If $deg_{G_{u_1}}(u_1) \leq 2$, Procedure Gen-$CS$ is invoked and $u_1$ is ejected. By the induction hypothesis, the invariant holds for $u_1$. Hence, $CS_{\delta(u_1)}$ can be used to generate a Mader construction sequence for $G_{\sigma(u_1)}$ based on Lemma 3.9.

If $deg_{G_{u_1}}(u_1) = 1$, $(w, u_1)$ is a bridge and $\hat{P}(u_1) = \bot$. The assertion vacuously holds true.

If $deg_{G_{u_1}}(u_1) = 2$ and $P_{u_1} : u_1$, then the cut pair is $\{(w \rightarrow u_1), (d \leftarrow u_1)\}$. Let $\{(w \rightarrow u_1), (d \leftarrow u_1)\}$ be the corresponding cut-pair in $G$ (note that $(d \leftarrow u_1)$ is an embodiment of $(d \leftarrow u_1)$). Clearly, $ear(w \rightarrow u_1) = (d \leftarrow u_1))$. Hence, $P_i(w) = P_{ear(w \rightarrow u_1)} = P_{(d \leftarrow u_1)}$. Since after $u_1$ is ejected, the path $q \rightarrow_{P_{(d \leftarrow u_1)}} w$ is replaced by the virtue edge $(d \leftarrow w)$, $(d \leftarrow w)$ thus becomes $P_i(w)$. Hence, we have $\hat{P}(u_1) = P_{(d \leftarrow w)} = (d \leftarrow w)$ and $P_{u_1} = nil$. Since $\hat{P}(w) = \bot$, the else-part of the third if statement is executed resulting in $\hat{P}(w) = (d \leftarrow w)$, $\overrightarrow{P}(w) = \bot$, $\delta(w) = \emptyset$, and $CS_{\delta(w)} = \bot$. The assertion thus holds.
If \( \text{deg}_{\hat{G}_{u_1}}(u_1) = 2 \) and \( \mathcal{P}_{u_1} : u_1 \rightarrow u_1^1 \ldots u_1^{h_1} \), then the cut pair is \( \{(w \rightarrow u_1), (u_1 \rightarrow u_1^1)\} \). After \( u_1 \) is ejected, \( \mathcal{P}_{u_1} : u_1^1 u_1^2 \ldots u_1^{h_1}(h_1 \geq 1) \). Since \( \hat{P}(u_1) = P_1(w) \) and \( \hat{P}(w) = \emptyset, \hat{P}(u_1) \preceq \hat{P}(w) \) and the else-part of the third if statement is executed resulting in \( \hat{P}(w) = P_1(w), \hat{P}(w) = \emptyset, \delta(w) = \emptyset, \) and \( \mathcal{C}S_{\delta(w)} = \emptyset. \) The assertion clearly holds.

(ii) If \( (u_1 \cap w) \in E \setminus E_T \), the argument is same as the above case when \( \text{deg}_{\hat{G}_{u_1}}(u_1) = 2 \) and \( \mathcal{P}_{u_1} : u_1, \) and after \( (d \cap w) \) becomes \( P_1(w). \) Hence, we have \( \hat{P}(w) = (u_1 \cap w), \hat{P}(w) = \emptyset, \delta(w) = \emptyset, \) \( \mathcal{C}S_{\delta(w)} = \emptyset, \) and the assertion holds.

Suppose the assertion holds for \( q - 1. \)

(i) If \( (w \rightarrow u_q) \in E_T. \) **Procedure** 3-edge-connect-\( \mathcal{C}S \) is invoked. When the dfs backtracks from \( u_q \) to \( w, \) if \( \text{deg}_{\hat{G}_{u_q}}(u_q) \leq 2, u_q \) is ejected and \( \mathcal{C}S_{\delta(u_q)} \) is used to generate a Mader construction sequence for \( \hat{G}_{\sigma(u_q)} \) based on Lemma 3.9. If \( \text{deg}_{\hat{G}_{u_q}}(u_q) = 1 \) (i.e. \( (w \rightarrow u_q) \) is a bridge), then \( P_q(w) = \emptyset \) and the assertion obviously holds for \( q. \) Otherwise, let \( \mathcal{P}_{u_q} : u_q^1 u_q^2 \ldots u_q^{h_q}(h_q \geq 1). \) Then, \( \delta_{u_q} = \{P_q(w)\} \cup \bigcup_{j=1}^{h_q} \delta(u_q^j) \) (note that \( \bigcup_{j=1}^{h_q} \delta(u_q^j) = \emptyset \) for the degenerated case \( P_q(w) = (d \cap w). \)) By assumption, the invariant holds for \( u_q^j, 1 \leq j \leq h_q, \) implying that \( \mathcal{C}S_{\delta(u_q^j)}, 1 \leq j \leq h_q \) are computed.

- If \( \hat{P}(w) \preceq \hat{P}(u_q)(= P_q(w)), \) then \( \hat{P}(w) = \min_{\mathcal{C}}\{P_i(w) \mid 1 \leq i \leq q - 1\} \) implies \( \hat{P}(w) = \min_{\mathcal{C}}\{P_i(w) \mid 1 \leq i \leq q\}. \) Moreover, **Procedure** Absorb-ear is invoked. On exiting the while loop, \( \sigma(w) = \sigma(w) \cup \bigcup_{j=1}^{h_q} \sigma(u_q^j) \) and as a result of absorbing \( \mathcal{P}_{u_q}, \delta(w) = \delta(w) \cup \delta_{u_q} = \bigcup_{i \leq q - 1 \wedge i \neq \ell} \delta_{u_i} \bigcup_{1 \leq i \leq q \wedge i \neq \ell} \delta_{u_q^j}, \) where \( \delta_{u_q} = \{P_{\text{ear}(w \rightarrow u_q)}\} \cup \bigcup_{j=1}^{h_q} \delta(u_q^j). \) By Lemma 3.7, \( \mathcal{C}S_{\delta(u_q)} = P_q(w) \mathcal{C}S_{\delta(u_q^1)} \mathcal{C}S_{\delta(u_q^2)} \ldots \mathcal{C}S_{\delta(u_q^{h_q})}. \)

If \( P_q(w) \preceq \hat{P}(w), \) then \( \hat{P}(w) = \min_{\mathcal{C}}\{P_i(w) \mid 1 \leq i \leq q - 1 \wedge i \neq \ell\} \) implies that \( P_q(w) = \min_{\mathcal{C}}\{P_i(w) \mid 1 \leq i \leq q \wedge i \neq \ell\} \) and the then-part of the if statement is executed making \( \hat{P}(w) = P_q(w). \) Moreover, By the induction hypothesis, \( \mathcal{C}S_{\delta(w)} = \mathcal{C}S_{\delta_{u_1}} \mathcal{C}S_{\delta_{u_1}} \ldots \mathcal{C}S_{\delta_{u_{(q-1)-2}}} \), where \( \{u_{i_1}, u_{i_2}, \ldots, u_{i_{(q-1)-2}}\} = \{u_1, u_2, \ldots, u_{q-1}\} \setminus \{u_{\ell}, u_{\ell}^j\}. \) Since after the ears in \( \mathcal{C}S_{\delta_{u_q}} \) are added to the construction sequence, \( w \) becomes a branch vertex. \( P_{\ell}(w) \) can thus be added to the construction sequence by operation (i) or (ii) following by the remaining ears in the current \( \mathcal{C}S_{\delta(w)}. \)

We thus have \( \mathcal{C}S_{\delta(w)} = \mathcal{C}S_{\delta_{u_q}} \mathcal{C}S_{\delta_{u_1}} \ldots \mathcal{C}S_{\delta_{u_{(q-1)-2}}} \). Since the anchor of \( \mathcal{C}S_{\delta_{u_q}} \) is \( P_q(w) = \hat{P}(w), \) the assertion thus holds.

If \( \hat{P}(w) \preceq P_q(w), \) the else-part is executed resulting in \( \hat{P}(w) = P_{\ell}(w) = \min_{\mathcal{C}}\{P_i(w) \mid 1 \leq i \leq q\}. \) \( \hat{P}(w) = P_{\ell}(w) = \min_{\mathcal{C}}\{P_i(w) \mid 1 \leq i \leq q \wedge i \neq q\}, \delta(w) = \bigcup_{\delta_{u_i} \mid 1 \leq i \leq q \wedge i \neq q} \delta(u_i), \) and \( \mathcal{C}S_{\delta(w)} = \mathcal{C}S_{\delta_{u_1}} \mathcal{C}S_{\delta_{u_1}} \ldots \mathcal{C}S_{\delta_{u_{(q-1)-2}}} \mathcal{C}S_{\delta_{u_q}} \) as after the ears in the current \( \mathcal{C}S_{\delta(w)} \) are added
to the construction sequence, $w$ becomes a branch vertex, and $P_q(w)$ can be added to the sequence with operation $(i)$ or $(ii)$ following by the remaining ears in $CS_{\delta u_q}$. Since the anchor of $CS_{\delta u_q}$ is $P_q(w) = \overrightarrow{P}(w)$, the assertion thus holds.

- If $\hat{P}(u_q) \leq \hat{P}(w)$, the argument is same as the above case except that the current $\mathcal{P}_u$ is absorbed by $w$.

As a result, $\hat{P}(w) = P_q(w) = \min_{\leq} \{ P_i(w) \mid 1 \leq i \leq q \}, \overrightarrow{P}(w) = P_{\ell}(w) = \min_{\leq} \{ P_i(w) \mid 1 \leq i \leq q \} \wedge i \neq q \}$, $\delta(w) = \bigcup \{ \delta_{u_i} \mid 1 \leq i \leq q \wedge i \neq q \}$, and $CS_{\delta(w)} = CS_{\delta u_q} CS_{\delta u_1} \ldots CS_{\delta u_{(q-1)-2}}$ as after the ears in $CS_{\delta u_q}$ are added to the construction sequence, $w$ becomes a branch vertex. $P_{\ell}(w)$ can thus be added to the sequence with operation $(i)$ or $(ii)$ following by the remaining ears in the current $CS_{\delta(w)} = CS_{\delta u_q} CS_{\delta u_1} \ldots CS_{\delta u_{(q-1)-2}}$. Since $P_{\ell}(w) = \overrightarrow{P}(w)$, the assertion thus holds.

(ii) If $(u_q \wedge w) \in E \setminus E_T$, the else-part of the if statement is executed. The remaining argument is same as the above case but much simpler as $\mathcal{P}_{u_q} = \text{nil}$, and $\delta_{u_q} = \{(u_q \wedge w)\}$ and $CS_{\delta_{u_q}} = (u_q \wedge w)$ by Lemma 3.7.

The assertion thus holds for $q$. Hence, when $L[w]$ is completely processed,

$\hat{P}(w) = P_{\ell}(w) = \min_{\leq} \{ P_i(w) \mid 1 \leq i \leq m \} = \min_{\leq} \hat{P}_w, \sigma(w) = \{ w \} \cup \bigcup_{j=1}^{h_i} \{ \sigma(u_i) \mid 1 \leq i \leq m \wedge i \neq \ell \}$ which consists of the vertices in $V_{T_w} \setminus V_{T_{\ell-1}}$ which are precisely the vertices absorbed by $w$,

$\delta(w) = \bigcup \{ \delta_{u_i} \mid 1 \leq i \leq m \wedge i \neq \ell \} = \bigcup \{ \{ P_i(w) \} \cup \bigcup_{j=1}^{h_i} \delta(u_i) \mid 1 \leq i \leq m \wedge i \neq \ell \} = \{ P_i(w) \mid 1 \leq i \leq m \wedge i \neq \ell \}$ which is the set of ears with their sink in $\sigma(w)$,

$\overrightarrow{P}(w) = P_{\ell}(w) = \min_{\leq} \{ \{ P_i(w) \mid 1 \leq i \leq m \wedge i \neq \ell \} = \min_{\leq} \{ \min_{\leq} \delta_{u_i} \mid 1 \leq i \leq m \wedge i \neq \ell \}$ (since $P_{\ell}(w) = \min_{\leq} \delta_{u_i} = \min_{\leq} (\bigcup \{ \delta_{u_i} \mid 1 \leq i \leq m \wedge i \neq \ell \}) = \min_{\leq} \delta(w)$, and

$CS_{\delta(w)} = CS_{\delta u_q} CS_{\delta u_1} \ldots CS_{\delta u_{m-2}}$, where $\{ u_1, u_2, \ldots , u_{m-2} \} = \{ u_1, u_2, \ldots , u_m \} \setminus \{ u_\ell, u_{\ell-1} \}$ such that $CS_{\delta u_q} = \overrightarrow{P}(w)CS_{\delta(u_{\ell-1})} CS_{\delta(u_{\ell-2})} \ldots CS_{\delta(u_1)}$ and

$CS_{\delta u_{ij}} = P_{ij}(w)CS_{\delta(u_{\ell})} \ldots CS_{\delta(u_{\ell-2})} \ldots CS_{\delta(u_{ij})}$, $1 \leq j \leq m-2$, where $\hat{u}_j = u_{ij}$ and $\hat{h}_j = h_{ij}$.

If $\mathcal{P}_w = w$ or $Inc_w = \emptyset$, the if statement following the for loop is not executed and the dfs backtracks to $v$ with $CS_{\delta(w)}$. Hence, the invariant holds for vertex $w$.

Otherwise, Procedure Absorb-path is invoked. Let $\mathcal{P}_w$ be $(w =) w_0 w_1 \ldots w_k (k \geq 1)$. The first for loop determines the largest index $h$ such that there is a back-edge $(w \wedge x) \in Inc$ of which $(w \wedge h)$ is an embodiment. Then $\overrightarrow{P}(w_{\ell}) = \min_{\leq} \{ \overrightarrow{P}(w_j) \mid 0 \leq j \leq h \}$ is computed. By assumption, the invariant holds for $w_i$, $0 \leq i \leq h$. Hence, by Lemma 3.5 the if statement and the for loop following it generates $CS_{\delta(w)}$. The invariant thus holds for vertex $w$. Since vertex $w$ has absorbed the path $(w =) w_0 w_1 \ldots w_h$, the last for loop updates $\sigma(w) = \bigcup_{j=0}^{h_i} \sigma(w_j)$ and shortens $P_w$ to $(w =) w_0 w_{h+1} \ldots w_k$. 

23
Finally, if \( w = r \), when execution of \textbf{Procedure} 3-edge-connect-CS\((r, \perp)\) terminates, the last instruction in \textbf{Algorithm} Certifying-3-edge-connectivity makes \( \hat{P}(r)CS_{\delta(r)} \) the Mader construction sequence of \( \hat{G}_{\sigma(r)} \) based on Lemma 3.9(i). The theorem thus follows.

A complete example is given in the Appendix (Figure 7).

To assure that the algorithm runs in linear-time, we maintain the following data-structures:

(i) An array \( parent[w], w \in V \setminus \{r\} \), such that \( parent[w] \) is the parent vertex of \( w \) in \( T \): this array allows an ear \( P_f \) of length \( k \) to be traced in \( O(k) \) time starting from its back-edge \( f \). (ii) An array \( ear[w], w \in V \setminus \{r\} \), such that \( ear[w] = ear(parent[w] \rightarrow w) \): this array allows the ear to which a tree edge belongs to be determined in \( O(1) \) time. Both arrays can be created in \( O(|V|) \) time during the depth-first search.

The adjacency lists, \( L[w], w \in V \), can clearly be constructed in \( O(|V| + |E|) \) time. \( Inc_w \) and the \( CS \)'s are represented by linked lists.

**Theorem 3.11.** \textbf{Algorithm} Certifying-3-edge-connectivity takes \( O(|V| + |E|) \) time.

**Proof:** \textbf{Algorithm} Certifying-3-edge-connectivity is an extension of \textbf{Algorithm} 3-edge-connectivity of [30]. The extension includes new instructions for generating \( CS_{\delta(w)}, \hat{P}(w), \hat{\hat{P}}(w), Inc_w \), \( w \in V \). Since \textbf{Algorithm} 3-edge-connectivity takes \( O(|V| + |E|) \) time [30], it suffices to show that the extension takes \( O(|V| + |E|) \) time.

In \textbf{Procedure} 3-edge-connect-CS (whose counterpart is \textbf{Procedure} 3-edge-connect in [30]), the new instructions for initialization clearly takes \( O(1) \) time. The instructions for generating \( Inc_w \), \( w \in V \), takes \( O(\sum_{w \in V} |Inc_w|) = O(|E| - |E_T|) \) time. \textbf{Procedure} Absorb-ear increases the time spent on absorbing the entire \( u \)-path or \( w \)-path by a constant factor. In \textbf{Procedure} Absorb-path, the for each statement takes \( O(h + |Inc_w|) \) time (note that verifying ancestor relation can be done in \( O(1) \) time [30]). Determining \( \hat{\hat{P}}(w) \) takes \( O(h) \) time. The for loops take \( O(h) \) time. The remaining new instructions takes \( O(1) \) time. Since each tree-edge is absorbed at most once, \( \sum_{w \in V} (O(h)) = O|E_T| \). Hence, in total, the procedure takes \( \sum_{w \in V} (O(h + |Inc_w|) + O(h) + O(1)) = O(|E_T|) + O(|E| - |E_T|) + O(|E_T|) + O(|V|) = O(|E|) \) time. \textbf{Procedure} Gen-CS clearly takes \( O(1) \) time. Since \( \hat{G}_{\sigma(w)}, w \in V \), are disjoint and each virtual edge in \( \hat{G}_{\sigma(w)} \) corresponds to a cut-edge in \( G \), generating their Mader construction sequences takes \( O(|V| + |E|) \) time. Hence, the new instructions take \( O(|V| + |E|) \) time. \textbf{Algorithm} Certifying-3-edge-connectivity thus runs in \( O(|V| + |E|) \) time. ■

24
3.3 Generating a cactus representation (negative certificate)

Let \( G = (V, E) \) be a 2-edge-connected undirected graph containing cut-pairs. A cactus representation of the cut-pairs of \( G \) consists of an undirected graph \( C_G = (V_C, E_C) \), where the elements in \( V_C \) are called nodes, and a function \( \Phi : V \rightarrow V_C \) such that the cut-pairs of \( G \) are precisely the preimages of the cut-pairs of \( C_G \). Specifically, \( \forall X \subseteq V, (X, \overline{X}) \) is a cut-pair in \( C_G \) if and only if \((\Phi^{-1}(X), \Phi^{-1}(\overline{X}))\) is a cut-pair in \( G \) (Note that \((X, \overline{X})\) is the pair of edges with one end-vertex in \( X \) and the other in \( \overline{X} \), where \( \overline{X} = V_C \setminus X \)). Nagamochi et al. [21] pointed out that the nodes in \( V_C \) are precisely the 3eccs of \( G \) which can be determined by contracting the latter. Since Algorithm 3-edge-connectivity uses contraction to determine the 3eccs, it can be easily modified to produce a cactus representation of the cut-pairs for each 2-edge-connected component of \( G \).

Since if \( \{e, e'\} \) and \( \{e', e''\} \) are cut-pairs, \( \{e, e''\} \) is also a cut-pair [28, 31], the set of all cut-edges can be partitioned into a collection of disjoint subsets \( E_i, 1 \leq i \leq \kappa \), such that every two edges from the same subset form a cut-pair and no two edges from different subsets form a cut-pair. Moreover, as the two edges of each cut-pair lie on the same ear [28, 31], the cut-edges in each \( E_i \) lie on the same ear \( P \) and hence can be lined up along \( P \) in an order \( e_1 \rightarrow e_2 \rightarrow \cdots \rightarrow e_{|E_i|} \), where \( e_1 = (x_1 \leftrightarrow y_1) \) or \( e_1 = (x_1 \rightarrow y_1) \), and \( e_j = (x_j \rightarrow y_j), 2 \leq j \leq |E_i| \), such that \( x_{j+1} \preceq y_j, 1 \leq j < |E_i| \). Note that we use \((x \leftrightarrow y)\) instead of \((y \lor x)\) to represent back-edges in this section and that the orientation of the tree edges on \( P \) follows that of \( P \) which is from the child to the parent. Each \( E_i \) is called a cut-edge chain and the cut-edge \( e_1 \) is called the generator of \( E_i \) [28, 31]. From the above discussion, each \( E_i \) gives rise to a unique cycle in the cactus \( C_G \). Hence, it remains to show how to convert \( E_i \) to a cactus cycle.

Let \( P \) be the ear containing \( E_i \). Then, either

(i) \( P : s(P) = x \xrightarrow{f} y \xrightarrow{X} x_1 \xrightarrow{e_1} y_1 \xrightarrow{T} x_2 \xrightarrow{e_2} y_2 \xrightarrow{T} \cdots \xrightarrow{T} x_{|E_i|} \xrightarrow{e_{|E_i|}} y_{|E_i|} \xrightarrow{T} t(P) \), or

(ii) \( P : s(P) = x_1 \xrightarrow{e_1} y_1 \xrightarrow{T} x_2 \xrightarrow{e_2} y_2 \xrightarrow{T} \cdots \xrightarrow{T} x_{|E_i|} \xrightarrow{e_{|E_i|}} y_{|E_i|} \xrightarrow{T} t(P) \).

Note that the generator \( e_1 \) is a tree-edge in Case (i) and a back-edge in Case (ii). We shall modify Algorithm 3-edge-connectivity to generate a cycle, \( \sigma(x_1)\sigma(x_2)\cdots\sigma(x_{|E_i|})\sigma(x_1) \), for \( C_G \) based on \( P \) and \( E_i \) as follows.

First, consider Case (i) (Figure 5(i)). When the dfs backtracks from \( x_2 \) to \( y_2 \), either \( x_2 = y_1 \) or \( x_2 \) has absorbed \( y_1 \) (directly or indirectly). Hence, \( P_{x_2} \) is \( x_2x_1v_1 \cdots v_h \). Since \( \deg_G(x_2) = 2 \), \( x_2 \) is ejected and \( \sigma(x_2) \) becomes a 3ecc which is a node in the cactus cycle corresponding to \( E_i \). This implies that the path \( \sigma(x_1)\sigma(x_2)\sigma(y_2) \) exists in \( C_G \). This path is represented by its internal node \( \sigma(x_2) \) which is attached
to $x_1$ as $x_1.t\text{chain}$. When the dfs backtracks from $x_3$ to $y_3$, either $x_3 = y_2$ or $x_3$ has absorbed $y_2$. $P_{x_3}$ is thus $x_3x_1v_1 \ldots v_h$ and $\sigma(x_3) = \sigma(y_2)$. Hence, $x_1.t\text{chain}$ represents the path $\sigma(x_1)\sigma(x_2)\sigma(x_3)$. Since $deg_G(x_3) = 2$, $x_3$ is ejected and $\sigma(x_3)$ becomes an internal node of the path $\sigma(x_1)\sigma(x_2)\sigma(x_3)\sigma(y_3)$ in $G$. Hence, $x_1.t\text{chain}$ becomes $\sigma(x_2)\sigma(x_3)$. Similarly, when the dfs backtracks from $x_{|\mathcal{E}_1|}$ to $y_{|\mathcal{E}_1|}$, $P_{x_{|\mathcal{E}_1|}}$ is $x_{|\mathcal{E}_1|}v_1 \ldots v_h$. Since $deg_G(x_{|\mathcal{E}_1|}) = 2$, $x_{|\mathcal{E}_1|}$ is ejected and $\sigma(x_{|\mathcal{E}_1|})$ becomes an internal node of the path $\sigma(x_1)\sigma(x_2)\sigma(x_3) \ldots \sigma(x_{|\mathcal{E}_1|})\sigma(y_{|\mathcal{E}_1|})$ in $G$. Hence, $x_1.t\text{chain}$ becomes $\sigma(x_2)\sigma(x_3) \ldots \sigma(x_{|\mathcal{E}_1|})$. Then, one of the following two cases follows.

(a) There exists an ear $P'$ such that $s(P') \leq y_{|\mathcal{E}_1|}$ and $x_1 \leq t(P')$: (Figure 5(i)(a)). Let $P_f$ be the one with $s(P_f)$ closest to $y_{|\mathcal{E}_1|}$. After the dfs backtracks to $s(P_f)$, when the back-edge $f$ is encountered, let $P_{s(P_f)}$ be $s(P_f)u_1u_2 \ldots u_{\ell}v_1 = v_0v_1v_2 \ldots v_h$, $(\ell, h \geq 0)$, (note that $y_{|\mathcal{E}_1|} \in \sigma(u_{\ell})$) and $t(P_f) \in \sigma(v_q)$. Then, $s(P_f)$ absorbs $u_1u_2 \ldots u_{\ell}v_1 \ldots v_h$ which includes $x_1$ and $u_{\ell}$ (hence, $y_{|\mathcal{E}_1|}$). As a result, $\sigma(s(P_f)) = \sigma(x_1) = \sigma(y_{|\mathcal{E}_1|})$. Hence, the path $\sigma(x_2)\sigma(x_3) \ldots \sigma(x_{|\mathcal{E}_1|})$ kept in $x_1.t\text{chain}$ and $\sigma(x_1)$ form a cactus cycle $\sigma(x_1)\sigma(x_2)\sigma(x_3) \ldots \sigma(x_{|\mathcal{E}_1|})\sigma(x_1)$ in $G$. The cycle is attached to $s(P_f)$. If $s(P_f)$ is absorbed by another vertex at a later stage, the cycle is attached to that vertex until a vertex $z$ to which the cycle is attached is ejected. Then, as $\sigma(z) = \sigma(x_1)$, $\sigma(z)\sigma(x_2)\sigma(x_3) \ldots \sigma(x_{|\mathcal{E}_1|})\sigma(z)$ is a cactus cycle in $G$.

(b) There does not exist such an ear $P'$: (Figure 5(i)(b)) After the dfs backtracks to $t(P)$, let $t(P) \rightarrow_T y$ be transformed into $t(P)u_1u_2 \ldots u_{\ell}v_0 = x_0v_1v_2 \ldots v_h$, $(\ell, h \geq 0)$, (note that $y_{|\mathcal{E}_1|} \in \sigma(u_{\ell})$) when $t(P)$
is to absorb it. Then, after \( t(P) \) absorbed \( x_1 \) and \( u_t \) (hence, \( y_{|E_i|} \)), \( \sigma(x_1) = \sigma(y_{|E_i|}) \) which implies that \( x_1.tchain \) and \( \sigma(x_1) \) form a cactus cycle \( \sigma(x_1)\sigma(x_2)\sigma(x_3)\ldots\sigma(x_{|E_i|})\sigma(x_1) \) in \( C_G \). The rest of the argument is same as that for Case \((a)\).

Next, consider Case \((ii)\) (Figure 5(ii)). This case is similar to Case \((i)(b)\) except after ejecting \( x_2 \), the node \( \sigma(x_2) \) is attached to \( y_2 \) instead of \( x_1 \) as \( y_2.bchain \). Then, \( y_2.bchain \) is transferred along \( y_2 \sim_T t(P) \) through the absorb operation until it is transferred to \( x_3 \), at which it becomes \( x_3.bchain \). After the \( dfs \) backtracks to \( y_3 \), \( x_3 \) is ejected and \( x_3.bchain \) is extended to \( \sigma(x_2)\sigma(x_3) \) which then becomes \( y_{|E_i|}.bchain \). This process is repeated for \( x_i, 4 \leq i \leq |E_i| \). When the \( dfs \) backtracks to \( y_{|E_i|} \), after \( x_{|E_i|} \) is ejected, \( x_{|E_i|}.bchain \) is extended to \( \sigma(x_2)\sigma(x_3)\ldots\sigma(x_{|E_i|}) \) which then becomes \( y_{|E_i|}.bchain \). After the \( dfs \) backtracks to \( t(P) \), when \( t(P) \) absorbs \( u_t \) such that \( y_{|E_i|} \in \sigma(u_t) \) and \( u_t.bchain = y_{|E_i|}.bchain, \sigma(t(P)) = \sigma(y_{|E_i|}) \) which implies that \( y_{|E_i|}.bchain \) represents the path \( \sigma(x_1)\sigma(x_2)\sigma(x_3)\ldots\sigma(x_{|E_i|})\sigma(t(P)) \). Since \( (x_1 =)s(P) \) and \( t(P) \) are 3-edge-connected, \( \sigma(x_1) = \sigma(t(P)) \). Hence, \( y_{|E_i|}.bchain \) and \( \sigma(x_1) \) form a cactus cycle \( \sigma(x_1)\sigma(x_2)\sigma(x_3)\ldots\sigma(x_{|E_i|})\sigma(x_1) \) in \( C_G \). The remaining argument is same as that for Case \((i)(a)\).

The above modifications can be easily incorporated into Algorithm Certifying-3-edge-connectivity. The following is a pseudo-code of the modified algorithm. For clarity, instructions that are not related to the construction of \( C_G \) have been omitted. The new instructions for generating the cactus are marked by ⋆.

At each vertex \( x \), the following variables are maintained:

- \( x.tchain \): a chain of nodes (\( \sigma(x_i) \)'s) attached to \( x \) corresponding to a cut-edge-chain whose generator is the parent edge of \( x \), and \( \sigma(x) + x.tchain \) is a path in \( C_G \);
- \( x.bchain \): a chain of nodes (\( \sigma(x_i) \)'s) attached to \( x \) corresponding to a cut-edge-chain with back-edge generator such that \( x.bchain + \sigma(x) \) is a path in \( C_G \);
- \( C.cycle(x) \): contains the cactus cycles attached to \( x \).

A node \( \sigma(x) \) in a cactus cycles is represented by \( x \). When a vertex \( u \) is ejected, if \( \deg_{G_u}(u) = 1 \), then a cactus representation of the cut-pairs in \( G_{\{\sigma(u)\}} \) is constructed. If \( \deg_{G_u}(u) = 2 \), \( \sigma(u) \) becomes a node in \( C_G \). Procedure Gen-CS is called to append \( \sigma(u) \) to \( u_1.tchain \) if both cut-edges are tree-edge, where \( u_1 \) is the vertex following \( u \) in the \( u \)-path before \( u \) is ejected, or to \( u.bchain \), otherwise. In the latter case, the updated \( u.bchain \) is temporary kept at \( bchain_t \) because if \( P_w = w, w.bchain \) might have contained a cactus chain. If \( P_u \ll P_w \), then \( bchain_t \) becomes \( w.bchain \) after the current \( w.bchain \) (if exists) is turned into a cactus cycle and attached to \( w \); if \( P_w \ll P_u \), \( bchain_t \) is turned into a cactus cycle and attached to \( w \) (Figure 6(i)). When vertex \( w \) absorbs \( P_w \) or \( P_u \), Procedure Absorb-ear is called to transfer all
cactus cycles attached to each absorbed vertex to \( w \), and convert the \( tchain \) (if exists) of each absorbed vertex as well as the \( bchain \) (if exists) of the last absorbed vertex into cactus cycles and attach them to \( w \) (Figure 6(ii)). **Procedure Absorb-path** is similar to **Procedure Absorb-ear** but is called to absorb a section of \( \mathcal{P}_w \) only. An example is given in the Appendix (Figure 8).

![Diagram](image)

**Figure 6**

**Algorithm Certifying-3-edge-connectivity**

**Input:** A connected graph \( G = (V, E) \) represented by adjacency lists \( L[w], \forall w \in V \)

**Output:**
- \( \mathcal{C}S_{\mathcal{L}(w)} ; u \in V \) Mader construction sequence for each \( u \) representing a 3ecc of \( G \),
- \( \{C_G' | G' \text{ is a 2ecc of } G\} \) a cactus representation of the cut-pairs for each 2ecc \( G' \) of \( G \), and
- \( \text{Bridges} \) the bridges in \( G \)

**begin**

for every \( u \in V \) do \( dfs(u) := 0 ; \) parent \((u) := \perp ; \) lowpt \((u) := \infty ; \) // initialization; \( \perp \) = undefined

- \( P(u) := \hat{P}(u) := \perp ; \) // note: \( \perp > P_f \forall f \in E \setminus E_T ; \)
- \( \sigma(u) := \{u\} ; \) Inc \(_u\) := \( 0 ; \) \( \mathcal{P}_u := u ; \)

- \( cnt := 1 ; \) // \( dfs \) number counter

**3-edge-connect-CS**(\( r, \perp \))

- for each (cycle \( xQx \in C\text{-cycle}(r) \land x \neq r \)) do
- Convert \( xQx \) to \( rQr ; \) // make \( r \) the starting and ending node of the cycle; note: \( \sigma(r) = \sigma(x) \)

**end.**

**Procedure 3-edge-connect-CS**(\( w, v \))

**begin**

- \( dfs(w) := cnt ; \) cnt := \( cnt + 1 ; \) parent \((w) := v ; \) lowpt \((w) := dfs(w) ; \)
- \( w\_tchain := w\_bchain := bchain_t := \perp ; \) C\_cycle\((w) := \emptyset ; \) // initialization

- for each \( u \in L[w] \) do
  - if \( (dfs(u) = 0) \) then
    - 3-edge-connect-CS\((u, w) ; \)
      - if \( (s(P(u)) = u \lor \hat{P}(u) = \perp) \) then
        - Gen-\(\mathcal{C}\)\((w, u, P_u, bchain_t) ; \)
      - else // \((P(u) \neq \hat{P}(u)) \)
        - Absorb-ear\((w, u, P(u), \mathcal{P}_u, bchain_t) ; \)
      - \( w\_bchain := bchain_t ; \) bchain := \( \perp ; \)
      - \( \mathcal{P}_w := w + \mathcal{P}_u ; \) \( \hat{P}(w) := \hat{P}(u) ; \)
    - // equivalent to \((deg_{G_u}(u) \leq 2) \)
    - // create a node \( \sigma(u) \) in the cactus
    - // equivalent to \((lowpt(w) \leq lowpt(u)) \) in \([30]\)
    - // \( w \) absorbs the entire \( w\_path \)
    - // equivalent to \((lowpt(w) > lowpt(u)) \) in \([30]\)
    - // \( w \) absorbs the entire \( w\_path \)
    - // transfer \( bchain_t \) to \( w\_bchain \)

- // \( u \) is unvisited

28
\begin{itemize}
  \item \texttt{Absorb-ear}(w, u, \hat{P}(w), P_w, \perp); \quad \text{if } (w \neq r) \text{ then } \texttt{ear}(v \rightarrow w) := \texttt{ear}(w \rightarrow u); \quad \text{else if } (dfs(u) < dfs(w) \land u \neq \text{parent}(u)) \text{ then } \text{else if } (dfs(u) < dfs(w) \land u \neq \text{parent}(u)) \text{ then } \end{itemize}

\texttt{Absorb-path}(w, \hat{P}(w), Inc_w); \quad \text{if } ((P_w \neq \perp) \land (Inc_w \neq \emptyset)) \text{ then } \texttt{Absorb-path}(w, P_w, Inc_w); \quad \text{if } (dfs(u) < dfs(w) \land u \neq \text{parent}(u)) \text{ then } \text{else if } (dfs(u) < dfs(w) \land u \neq \text{parent}(u)) \text{ then }
For each cut-edge chain \( E : e_1 e_2 \ldots e_{|E|} \), where 
\( e_1 = (x_1 \leftrightarrow y_1) \) or 
\( (x_1 \rightarrow y_1) \), 
\( e_i = (x_i \rightarrow y_i) ; y_i < x_i \leq y_{i-1} \), 
\( 2 \leq i \leq |E| \), 
such that \( w \leq y_2 \), either

(a) there exists a cactus cycle \( \sigma(x_1) \ldots \sigma(x_{|E|}) \sigma(x_1) \) attached to a vertex \( x \) such that \( \sigma(x) = \sigma(x_1) \), where \( x \) is some \( w_j, 0 \leq j \leq k \), or \( x_1 = x \) and \( x \) has been ejected by the eject-absorb operation; or

(b) there exists a path \( \sigma(x_2) \ldots \sigma(x_\ell) \), where \( w \prec x_\ell \wedge (x_{\ell+1} \leq w \lor \ell = |E|) \), attached to some \( w_j, 1 \leq j \leq k \), as \( w_j.tchain \) such that \( w_j = x_1 \) if the generator \( e_1 \) is a tree-edge, or attached to \( w_k \) as \( w_k.bchain \) such that \( \sigma(w_k) = \sigma(y_\ell) \) if the generator \( e_1 \) is a back-edge.

**Proof:** (By induction on the height of \( w \) in \( T \))

If \( w \) is a leaf, the assertion vacuously holds true.

Let \( w \) be an internal vertex. When the \texttt{dfs} backtracks from a child \( u \) to \( w \), if \( \text{deg}_{G_u}(u) \leq 2 \), **Procedure Gen-CS** is invoked. By the induction hypothesis, the assertion holds for \( u \). By (a), each cut-edge chain in \( G_{\sigma(u)} \) has been transformed to a cactus cycle \( xQx \) \((Q \text{ is a path})\) which is attached to \( u \) or to a vertex \( x \) that has been rejected. Then for those cycles \( xQx \) attached to \( u \) such that \( x \neq u \), since \( \sigma(x) = \sigma(u) \), the for loop replaces the starting and ending vertex \( x \) with \( u \), resulting in \( uQu \). Then, if \( \text{deg}_{G_u}(u) = 1 \), a cactus representation of the cut-pairs in \( G_{\sigma(u)} \) is created.

If \( \text{deg}_{G_u}(u) = 2 \), then either \( \{(w \rightarrow u), (u \rightarrow u_1)\} \) or \( \{(w \rightarrow u), (d \land u)\} \) is a cut-pair.

In the former case, \( (u \rightarrow w) \) is a cut-edge of a cut-edge chain \( E : e_1 e_2 \ldots e_{\ell} \ldots e_{|E|} \) such that 
\( e_i = (x_i \rightarrow y_i) ; 1 \leq i \leq |E| \), and 
\( e_\ell = (x_\ell \rightarrow y_\ell) = (u \rightarrow w) \). Then \( u_1 = x_1 \) and \( u_1.tchain = \sigma(x_2) \sigma(x_3) \ldots \sigma(x_{\ell-1}) \). After executing the else part of the second if statement, \( u_1.tchain = \sigma(x_2) \sigma(x_3) \ldots \sigma(x_\ell) \).

**Procedure Absorb-ear** is then invoked. • If \( \hat{P}(w) \ll \hat{P}(u) \), \( \hat{P}(u) \) terminates at \( w \). Hence, \( w \) absorbs \( \mathcal{P}_u : u_1 \ldots u_h (h \geq 1) \) resulting in \( \sigma(w) = \sigma(u_i) ; 1 \leq i \leq h \), and the cactus cycle attached to \( u_i \) are transferred to \( w \). Moreover, as \( \sigma(u_i) = \sigma(u_{i-1}) ; 1 < i \leq h \), If \( u_i.tchain \) is not null, \( u_i.tchain \) and \( \sigma(w) \) form a cactus cycle. The cycle \( w + u_i.tchain + w \) is thus created and attached to \( w \). If \( w_h.bchain \) is not null, it is also turned into a cactus cycle \( w + w_h.bchain + w \) and attached to \( w \). • If \( \hat{P}(u) \ll \hat{P}(w) \), \( \hat{P}(w) \) terminates at \( w \), and vertex \( w \) absorbs the \( w \)-path \( \mathcal{P}_w : w_1 \ldots w_k \). The process is same as that for absorbing \( \mathcal{P}_u \) above.

In the latter case, \( (u \rightarrow w) \) is a cut-edge of a cut-edge chain \( E : e_1 e_2 \ldots e_{\ell} \ldots e_{|E|} \) such that 
\( e_1 = (x_1 \leftrightarrow y_1) \), 
\( e_i = (x_i \rightarrow y_i) ; 2 \leq i \leq |E| \), and 
\( e_\ell = (x_\ell \rightarrow y_\ell) = (u \rightarrow w) \). Since (b) holds for \( u \), 
\( u.bchain = \sigma(x_2) \sigma(x_3) \ldots \sigma(x_{\ell-1}) \). After executing the then part of the second if statement, \( u.bchain + \sigma(x_\ell) = \sigma(x_2) \sigma(x_3) \ldots \sigma(x_\ell) \) is created which is stored at \( bchain_\ell \). **Procedure Absorb-ear** is then invoked. • If \( \hat{P}(w) \ll \hat{P}(u) \), since \( bchain_\ell \) is not null, the then part of the first if statement is executed and
the cycle \( w + \text{bchain}_t + w \) is created and attached to \( w \). • If \( \hat{P}(u) \leq \hat{P}(w) \), \( \hat{P}(w) \) terminates at \( w \). Hence, vertex \( w \) absorbs the \( w \)-path \( \mathcal{P}_w : w_1 \ldots w_k \). The process is same as that for absorbing \( \mathcal{P}_u \) above. After that \( \text{bchain}_t \) is transferred to \( w.\text{bchain} \).

If \( \deg_{G_u}(u) > 2 \), then by Lemma[3,3(ii)], there exists \( (x \sim u) \) with \( x \preceq w \) which implies that \( (w \rightarrow u) \) does not form a cut-pair with an edge on \( \mathcal{P}_u \). Hence, \( u.tchain = \bot \) and \( u.\text{bchain} \) remains unchanged. The remaining argument is same as the above case.

When an \( (u \sim w) \) is encountered, the argument is same as the above case when \( u \) is a child of \( w \) but much simpler as \( \mathcal{P}_u \) is \text{nil} and \( \text{bchain}_t \) is not involved.

Hence, when \( L[w] \) is completely processed, both \( (a) \) and \( (b) \) hold for \( w \).

Finally, if \( (\mathcal{P}_w \neq \text{nil}) \land (\text{Inc}_w \neq \emptyset) \), then \textbf{Procedure Absorb-path} is invoked and vertex \( w \) absorbs \( w_i, 1 \leq i \leq h \), on \( \mathcal{P}_w \). The process is same as that of absorbing the entire \( \mathcal{P}_w \) except that \( w_k.\text{bchain} \) is not involved if \( h = k \). Hence, when the \text{dfs} backtracks from \( w \), the assertion holds for vertex \( w \). \( \blacksquare \)

**Theorem 3.13. Algorithm Certifying 3-edge-connectedness generates a cactus representation of the cut-pairs for each 2-edge-connected components of \( G \).**

**Proof:** Let the parent edge of \( w \) be a bridge. Then \( \deg_{G_w}(w) = 1 \). By Lemma[3,12], when the \text{dfs} backtracks from \( w \) to its parent, Condition \( (a) \) or \( (b) \) holds for every cut-pair chain with \( w \preceq y_2 \). Since \( \deg_{G_w}(w) = 1 \), \( \mathcal{P}_w = w \) which implies that \( k = 0 \). Hence, only \( (a) \) holds for every cut-pair chain in \( G_{(\sigma(w))} \). After the \textbf{for} loop in \textbf{Procedure Gen-CS} relabels the starting and ending vertex of each cactus cycle attached to \( w \) with \( \sigma(w) \), a cactus representation of the cut-edges in \( G_{(\sigma(w))} \) is constructed.

For the root \( r \), when execution of \textbf{Procedure 3-edge-connect-CS}(r, \bot) terminates, \( \mathcal{P}_r = r \) which implies that only \( (a) \) holds for every cut-pair chain in \( G_{(\sigma(r))} \). Hence, after the last \textbf{for} loop relabels the starting and ending vertex of each cactus cycle attached to \( r \) with \( \sigma(r) \), a cactus representation of the cut-edges in \( G_{(\sigma(r))} \) is constructed. \( \blacksquare \)

**Theorem 3.14. Procedure 3-edge-connect-CS runs in \( O(|V| + |E|) \) time.**

**Proof:** The time complexity follows from the fact that the new instructions increase the run time of Procedures 3-edge-connect-CS, Absorb-ear and Absorb-path by a constant factor, and the total time spent on renaming the starting and ending nodes of the cactus cycles in \textbf{Procedure Gen-CS} is bounded by the total size of the cactus cycles which is clearly \( O(|E|) \). \( \blacksquare \)

The two algorithms presented can be easily combined into one so that the resulting \textbf{Algorithm Certifying-3-edge-connectedness} generates the 3eccs, the Mader construction sequences for the 3eccs, the bridges,
and a cactus representation of the cut-pairs for each of the 2-edge connected components of \( G \) simultaneously in \( O(|V| + |E|) \) time by making one pass over the input graph.

### 4 Conclusion

We presented a linear-time certifying algorithm for recognizing 3-edge-connected graphs. The algorithm does not require the input graph \( G \) to be 2-edge-connected and makes only one pass over \( G \) to seamlessly generate the following outputs: the 3\textit{eccs} of \( G \) each of which is accompanied by a Mader construction sequence serving as a positive certificate, a cactus representation of the cut-pairs of each 2-edge-connected component of \( G \), and the bridges. Clearly, if \( G \) is 3-edge-connected, only one Mader construction sequence is generated and no cactus representations nor bridges are generated. To verify the certificates, the methods of Mehlhorn et al. [19] can be used.

In [23], it is reported that to check the condition ‘\( \text{deg}_{G_u}(u) = 2 \)’, it is more efficient (in terms of execution time and implementation) to compute the degrees of the vertices directly than to maintain, at each vertex, a list of embodiments of the ears absorbed by the vertex [30]. In our algorithm, as we must compute \( \tilde{P}(u) \) for each vertex \( u \), and ‘\( \text{deg}^c_{G_u}(u) \leq 2 \)’ if and only if \( (\tilde{P}(u) = \perp) \lor (s(\tilde{P}(u)) = u) \)’, computing the degrees of the vertices is unnecessary.

### References

[1] Chin F., Chrobak M., Li Yan, “Algorithms for placing monitors in a flow network”, Algorithmica, vol. 68(1), 2014, 1-15.
[2] Cong, G., IBM TJ Watson Research Center, Yorktown Heights, NY, USA, Electronic Communications, 2011.
[3] Corcoran J.N., Schneider U., Schttler H.-B., “Perfect stochastic summation in high order Feynman graph expansions”, Int. J. Mod. Phys. C, vol. 17 (11), 2006, 1527-1549.
[4] Corneil D.G., Dalton B., Habib M., “LDFS based certifying algorithm for the Minimum Path Cover problem on cocomparability graphs,” SIAM J. on Comput., vol.42(3), 2013, 792-807.
[5] Dehne F., Langston M. A., Luo X., Pitre S., Shaw P., Zhang Y., “The cluster editing problem: implementations and experiments”, IWPEC, Zuerich, Switzerland, 2006, LNCS 4169, 13-24.
[6] Elmasry A., Mehlhorn K., Schmidt J.M., “An O(n+m) Certifying Triconnectivity Algorithm for Hamiltonian Graphs”, Algorithmica vol.62(3-4), 2012, 754-766.
[7] Even, S.: Graph Algorithms. Computer Science Press, Potomac, MD, 1979.
[8] Gabow H.N., “Path-based depth-first search for strong and biconnected components,” Information Processing Letters, vol. 74, 2000, 107-114.
[9] Galil Z., Italiano G.F., “Reducing edge connectivity to vertex connectivity”, SIGACT News 22, 1991, 57-61.
[10] Heggernes P., Kratsch D., “Linear-time certifying recognition algorithms and forbidden induced subgraphs,” Nordic J. of Computing 14(1), 2007, 87-108.
[11] Hell P., Huang J., “Certifying LexBFS recognition algorithms for proper interval graphs and proper interval bigraphs,” SIAM J. Discret. Math. 18(3), 2005, 554-570.
[12] Kaplan H., Nussbaum Y., “Certifying algorithms for recognizing proper circular-arc graphs and unit circular-arc graphs,” WG2006, LNCS vol. 4271, Springer, 2006, pp. 289-300.
[13] Kratsch D., McConnell R., Mehlhorn K., Spinrad J., “Certifying algorithms for recognizing interval graphs and permutation graphs,” SIAM Journal on Computing 36(2), 2006, 326-353.
[14] Liu V.Y., Tipper D., “Spare capacity allocation using shared backup protection for dual link failures”, Computer Communications, vol. 36, 2013, 666-677.
[15] Mader W., “A reduction for edge-connectivity in graphs”, In B.Bollobas (editor), Advances in Graph Theory, vol.3 of Annuals of Discrete Mathematics, 1978, 145-164.
[16] McConnell R.M., Mehlhorn K., Naher S., and Schweitzer P., “Certifying algorithms,” Computer Science Review vol.5, 119-161 (2011).
[17] Mehlhorn K., and Naher S., “From algorithms to working programs: On the use of program checking in LEDA,” Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS’98), 1998, 84-93.
[18] Mehlhorn K., and Naher S., The LEDA Platform for Combinatorial and Geometric Computing, Cambridge University Press, 1999.
[19] Mehlhorn K., Neumann A. and Schmidt J. M., “Certifying 3-edge connectivity”, Algorithmica vol. 77(2), 2017, 309-335.
[20] Nagamochi H., Ibaraki T., “A linear-time algorithm for computing 3-edge-connected components in a multigraph”, Japan J. Indust. Appl. Math., vol. 9, 1992, 163-180.
[21] Nagamochi H., Ibaraki T.: Algorithmic Aspects of Graph Connectivity (Encyclopedia of Mathematics and its Applications), Cambridge University Press, 2008.
[22] Nikolopoulos S.D., Palios L., “An O(nm)-time certifying algorithm for recognizing hhd-free graphs,” FAW 2007, LNCS vol. 4613, Springer, 2007, 281-292.
[23] Norouzi N., Tsin Y.H., “A simple 3-edge connected component algorithm revisited,” Information Processing Letters, vol. 114(1-2), 2014, 50-55.
[24] Olariu S., Zomaya A. Y., “A time- and cost-optimal algorithm for interlocking sets - with applications”, IEEE Trans. Parallel Distrib. Syst., vol. 7(10), 1996, 1009-1025.
[25] Paten B., Diekhans M., Earl D., St. John J., Suh B., Haussler D., “Cactus graphs for genome comparisons”, RECOMB, Lisbon, Portugal, 2010, LNCS 6044, 410-425.
[26] Schmidt J.M., “Contractions, Removals and Certifying 3-connectivity in linear time”, SIAM J. Comput., vol. 42(2), 2013, 494-535.
[27] Schmidt, J.M.: “A simple test on 2-vertex- and 2-edge-connectivity,” Information Processing Letters, vol. 113(7), 2013, 241-244.
[28] Taoka S., Watanabe T., Onaga K., “A linear-time algorithm for computing all 3-edge-connected components of a multigraph”, IEICE Trans. Fundamentals, vol. E75(3), 1992, 410-424.
[29] Tarjan R., “Depth-first search and linear graph algorithms”, SIAM J. Comput., vol. 1(2), 1972, 146-159.
[30] Tsin, Y.H., “A simple 3-edge-connected component algorithm”, Theory of Computing Systems, vol. 40(2), 2007, 125-142.
[31] Tsin, Y.H., “Yet another optimal algorithm for 3-edge-connectivity”, Journal of Discrete Algorithms, vol. 7(1), 2009, 130-146.
[32] Tsin Y.H., Recognizing and embedding outerplanar distributed computer networks, CyberC 2011, IEEE, Beijing, China, Oct 10-12, 2011, 212-219.
(i) A dfs starts from $v_1$ and traverses to $v_{13}$ at which: $\hat{P}(v_{13}) = P_1$; $\mathcal{CS}_{\delta(v_{13})} = P_2$ (note: for clarity, we use $P_i$ to denote $P_{f_i}$ and label the back-edge $f_i$ with $P_i$; we underline $\overrightarrow{P}(w)$ in $\mathcal{CS}_{\delta(w)}$ which is always the first ear).

(ii) dfs backtracks to $v_{12}$ and advances to $v_{14}$ at which: $\hat{P}(v_{14}) = P_4$; $\mathcal{CS}_{\delta(v_{14})} = P_3$.

(iii) dfs backups to $v_{12}$; as $P_1 \ll P_4$, $v_{12}$ absorbs $v_{14}$ giving $\mathcal{CS}_{\delta(v_{12})} = P_4P_3$; $\hat{P}(v_{12}) = P_1$.

(iv) dfs backups to $v_{11}$; as $w_e = w_h = v_{12}$, $v_{11}$ absorbs $v_{12}$ giving $\mathcal{CS}_{\delta(v_{11})} = P_4P_3$, $\hat{P}(v_{11}) = P_1$.

(v) dfs backups to $v_{10}$: as $w_e = w_h = v_{13}$, $v_{10}$ absorbs $v_{11}$, $v_{13}$, giving $\mathcal{CS}_{\delta(v_{10})} = P_2P_4P_3$; $\hat{P}(v_{10}) = P_1$.

(vi) dfs backups to $v_9$: As $\deg_{\tilde{G}_{v_{10}}}(v_{10}) = 1$, $(v_9, v_{10})$ is a bridge and $v_{10}$ is ejected. As $\hat{P}(v_{10}) = P_1$, $\hat{P}(v_{10})\mathcal{CS}_{\delta(v_{10})} = P_1P_2P_4P_3$ is a construction sequence of $\tilde{G}_{\delta(v_{10})}$. 

Figure 7
(vii) At \( v_9 \), \( CS_{\delta}(v_9) = P_8 \), \( \bar{P}(v_9) = P_5 \). The dfs backups to \( v_9 \): \( CS_{\delta}(v_9) = P_8P_7 \), \( \bar{P}(v_9) = P_5 \). (viii) dfs backups to \( v_7 \) and then advances to \( v_{22} \): \( \bar{P}(v_{22}) = P_{10}; CS_{\delta}(v_{22}) = P_9P_{11} \). (ix) dfs backups to \( v_{21} \) and then advances to \( v_{24} \): \( \bar{P}(v_{24}) = P_{13}; CS_{\delta}(v_{24}) = P_9 \). (x) dfs backups to \( v_{23} \): \( \bar{P}(v_{23}) = P_{13}; CS_{\delta}(v_{23}) = P_{14} \); then backups to \( v_{21} \) as \( P_{13} \prec P_{10}, v_{21} \) absorbs \( v_{22} \) giving \( CS_{\delta}(v_{21}) = P_9P_{10}P_{11}\); \( \bar{P}(v_{21}) = P_{13} \). (xi) dfs backups to \( v_{20} \) and advances to \( v_{26} \): \( \bar{P}(v_{26}) = P_{15}; CS_{\delta}(v_{26}) = P_{16} \); dfs backups to \( v_{25} \): \( \bar{P}(v_{25}) = P_{15}; CS_{\delta}(v_{25}) = P_{17} \). (xii) dfs backups to \( v_{20} \): as \( P_{13} \prec P_{15}, v_{20} \) absorbs \( v_{25} \), \( v_{26} \), giving \( CS_{\delta}(v_{20}) = P_{15}P_{16}P_{17}; \bar{P}(v_{20}) = P_{13} \). (xiii) dfs backups to \( v_{19} \): as \( w_{\ell} = w_h = v_{20}, v_{19} \) absorbs \( v_{20} \) giving \( CS_{\delta}(v_{19}) = P_{15}P_{16}P_{17}; \bar{P}(v_{19}) = P_{13} \). Then dfs backups to \( v_{18} \): as \( w_{\ell} = w_h = v_{19}, v_{18} \) absorbs \( v_{19} \) giving \( CS_{\delta}(v_{18}) = P_{15}P_{16}P_{17}; \bar{P}(v_{18}) = P_{13} \). (xiv) The dfs backups to \( v_{17} \): as \( deg_{G_{\delta}}(v_{18}) = 2, v_{18} \) is ejected; as \( u = v_{17} \), \( u_{\ell} = parent(v_{21}) = v_{20}, (v_{18} \sim T v_{20}) \bar{P}(v_{18}, v_{20})P_{15}P_{16}P_{17} \) is a construction sequence of \( \tilde{G}_{\delta}(v_{18}) \); a new (virtual) edge \( (v_{17} \rightarrow v_{21}) \) is added to the remaining graph, and \( \bar{P}(v_{17}) = P_{13} \). (xv) dfs advances to \( v_{27} \): \( \bar{P}(v_{27}) = P_{19}, CS_{\delta}(v_{27}) = P_{18} \).
(xvi) dfs backtracks to \( v_{17} \): as \( P_{13} \not< P_{19} \), \( v_{17} \) absorbs \( v_{27} \) giving \( CS_{\delta(v_{17})} = P_{19}P_{18} \) and \( \hat{P}(v_{17}) = P_{13} \).

(xvii) dfs backtracks to \( v_{16} \): first, \( (v_5 \leftarrow v_{16}) \) is processed giving \( CS_{\delta(v_6)} = P_{20} \).

(xviii) Then, the incoming back-edge \( (v_{16} \leftarrow v_{17}) \) is processed. As \( w_h = v_{17}, w_\ell = v_{16} \), \( v_{16} \) absorbs \( v_{17} \) giving \( CS_{\delta(v_{16})} = P_{20}P_{19}P_{18} \); \( \hat{P}(v_{16}) = P_{13} \).

(xix) dfs backtracks to \( v_{15} \): as \( P_9 \not< P_{13} \), \( v_7 \) absorbs the \( v_{15} \)-path: \( v_{15}v_{21}v_{23}v_{24} \), giving \( CS_{\delta(v_7)} = P(v_{15})CS_{\delta(v_{24})}CS_{\delta(v_{23})}CS_{\delta(v_{22})}CS_{\delta(v_{21})}CS_{\delta(v_{15})} = P_{19}P_{12}P_{14}P_{10}P_9P_{11}P_{20}P_{19}P_{18} \) and \( \hat{P}(v_7) = P_{13} \).

(xx) dfs backtracks to \( v_6 \): as \( P_8 = \min \{ P_6, P_5, P_{13} \} \), \( v_7 = v_8 = w_h = v_6 \), and \( v_6 \) absorbs the path: \( v_7v_8v_9 \) giving \( CS_{\delta(v_6)} = CS_{\delta(v_9)}CS_{\delta(v_8)}CS_{\delta(v_7)} \), \( CS_{\delta(v_6)} = P_8P_7P_9P_{13}P_{12}P_{14}P_{10}P_9P_{11}P_{20}P_{19}P_{18} \) and \( \hat{P}(v_6) = P_5 \).

(xxi) dfs backtracks to \( v_5 \): as \( w_h = w_\ell = v_5 \), \( v_5 \) absorbs \( v_6 \) giving \( CS_{\delta(v_5)} = P_8P_7P_6P_{13}P_{12}P_{14}P_{10}P_9P_{11}P_{20}P_{19}P_{18} \) and \( \hat{P}(v_5) = P_5 \).

(xxii) dfs backtracks to \( v_4 \): as \( 9 \mid G_{v_3} \mid (v_5) = 2, v_9 \) is ejected. Since \( w = v_5 \), and \( \hat{P}(v_5) = P_5 \), \( P_{e_{r}(v_4 \rightarrow v_5)} = P_{v_3 \leftarrow v_9} \) implies that \( \hat{w} = t(v_4 \leftarrow v_9) = v_9, (v_5 \leftarrow v_9)(v_5, v_9)P_8P_7P_6P_{13}P_{12}P_{14}P_{10}P_9P_{11}P_{20}P_{19}P_{18} \) is a construction sequence of \( G_{\delta(v_5)} \). A new (virtual) edge \( P'_5 = (v_1 \leftarrow v_4) \) is added to the remaining graph and \( \hat{P}(v_4) = P'_5 \).

(xxiv) dfs advances to \( v_{30} \) during which the back-edges (trivial ears) \( P_{21}, P_{22}, P_{23} \) are encountered in the listed order. Since \( P_{23} \not< P_{22} \), \( \hat{P}(v_{30}) = P_{23}, CS_{\delta(v_{30})} = P_{22} \).

(xxv) dfs backtracks to \( v_{29} \) at which the back-edge (trivial ears) \( P_{24} \) is encountered. Since \( P_{23} \not< P_{24} \), \( \hat{P}(v_{29}) = P_{23}, CS_{\delta(v_{29})} = P_{21} \).
(xxvi) \( \text{dfs backtracks to } v_{28}: \text{ Since } P_{23} \preceq P_{21}, \hat{P}(v_{28}) = P_{23}, CS_{\delta(v_{28})} = P_{23}. \)

(xxvii) \( \text{Then, as } P_{21} = \min_{v} \{P_{21}, P_{22}, P_{24}\}, w_{f} = v_{28}, w_{h} = v_{30}, \text{ and } \hat{v}_{28} \text{ absorbs the } v_{28}\text{-path: } v_{28}v_{29}v_{30} \) giving \( CS_{\delta(v_{28})} = P_{23}P_{22}P_{24} \) and \( \hat{P}(v_{28}) = P_{23}. \)

(xxviii) \( \text{dfs backups to } v_{27}: \text{ As } w_{f} = w_{h} = v_{28}, v_{27} \) absorbs the \( v_{27}\text{-path: } v_{27}v_{28} \) giving \( CS_{\delta(v_{27})} = P_{21}P_{22}P_{24} \) and \( \hat{P}(v_{27}) = P_{23}. \)

(xxix) \( \text{The } \text{dfs backups to } v_{4}: \text{ as } \hat{P}(v_{4}) = P_{5}' \preceq P_{23}(= \hat{P}(v_{27})), v_{4} \text{ absorbs } v_{27} \text{ giving } CS_{\delta(v_{4})} = \hat{P}(v_{27})CS_{\delta(v_{27})} = P_{23}P_{21}P_{22}P_{24}. \)

(XXX) \( \text{dfs backups to } v_{3}: \text{ as } deg_{\hat{G}_{\delta(v_{4})}}(v_{4}) = 2, v_{4} \) is ejected. As \( u = \hat{u} = v_{4}, CS_{\delta(v_{4})} = P_{23}P_{21}P_{22}P_{24} \) is a construction sequence of \( \hat{G}_{\delta(v_{4})}. \) A new edge \( P_{5}'' = (v_{1} \prec v_{3}) \) is added to the remaining graph, and \( \hat{P}(v_{3}) = P_{5}'' \).

(xxi) \( \text{dfs advances to } v_{32} \text{ at which } P_{25}, P_{26} \) are encountered: \( \hat{P}(v_{32}) = P_{26}, CS_{\delta(v_{32})} = P_{25}. \)

(xxii) \( \text{dfs backups to } v_{31} \text{ at which } \hat{P}(v_{31}) = P_{26} \) and then advances to \( v_{33} \) at which \( P_{27}, P_{28} \) are encountered: \( \hat{P}(v_{33}) = P_{27}, CS_{\delta(v_{33})} = P_{28}. \)

(xxiii) \( \text{dfs backups to } v_{31}: \text{ as } \hat{P}(v_{31}) = P_{26} \preceq P_{27}(= \hat{P}(v_{33})), v_{31} \text{ absorbs } v_{33} \text{ giving } CS_{\delta(v_{31})} = \hat{P}(v_{33})CS_{\delta(v_{33})} = P_{27}P_{28}, \) and \( \hat{P}(v_{31}) = P_{26}. \)

(xxiv) \( \text{dfs backups to } v_{3}: \text{ as } \hat{P}(v_{3}) = P_{26} \preceq P_{27}(= \hat{P}(v_{33})), CS_{\delta(v_{3})} = P_{27}'' \) and \( \hat{P}(v_{3}) = P_{26}. \)

(xxv) \( \text{dfs backups to } v_{2}: \text{ as } P_{27} = \min_{v} \{P_{25}, P_{27}, P_{5}''\}, v_{f} = v_{31}, v_{h} = v_{32}, \) and \( v_{2} \) absorbs the path \( v_{2}v_{3}v_{31}v_{32} \) giving \( CS_{\delta(v_{2})} = P_{27}P_{28}P_{29}P_{5}'' \), and \( \hat{P}(v_{2}) = P_{26}. \)

(xxvi) \( \text{dfs backups to } v_{1}: \hat{P}(v_{1}) = P_{26}. \) As \( w_{h} = v_{3} = v_{2}, v_{1} \) absorbs the path \( v_{1}v_{2} \) giving \( CS_{\delta(v_{1})} = P_{26}P_{27}P_{28}P_{29}P_{5}'' \) is a Mader construction sequence of \( \hat{G}_{\delta(v_{1})}. \)
(i), (ii) \( \text{dfs} \) starts from \( a \), advances to \( d \), then backtracks to \( c \) and advances to \( i \).

(iii) \( \text{dfs} \) backtracks from \( i \) to \( h \): eject \( i \), \( h.bchain:\{i\} \) (note: \( \{ \} \) denotes \( \sigma(\)\)

(iv) \( \text{dfs} \) backtracks to \( g \): eject \( h \), \( g.bchain:\{i\}-\{h\} \).

(v) \( \text{dfs} \) advances to \( n \).

(vi) \( \text{dfs} \) backtracks to \( k \): eject \( \ell \), \( m.tchain:\{\ell\} \).

(vii), (viii) \( \text{dfs} \) advances to \( o \), then backtracks to \( k \): eject \( o \), \( k.bchain:\{o\} \).

(ix) Vertex \( k \) absorbs \( m \)-path, \( \sigma(k) = \{k, m, n\} \), cycle \( \{k\}-\{\ell\}-\{k\} \) attached to \( k \).

(x) \( \text{dfs} \) backtracks to \( j \), \( j \) absorbs \( k \), \( \sigma(j) = \{j, k, m, n\} \), attach cycle \( \{k\}-\{\ell\}-\{k\} \) to \( j \), \( j.bchain:\{o\} \).

(xi) \( \text{dfs} \) backtracks to \( g \): eject \( j \), rename cycle \( \{k\}-\{\ell\}-\{k\} \) attached to \( j \) as \( \{j\}-\{\ell\}-\{j\} \), \( behain_j: \{o\}-\{j\} \).

Since \( \hat{P}(j) \preceq \hat{P}(g) \), convert \( g.bchain \) into cycle \( \{g\}-\{i\}-\{h\}-\{g\} \), transfer \( behain_j \) to \( g.bchain: \{o\}-\{j\} \).

(xii) \( \text{dfs} \) backtracks to \( f \): \( f \) absorbs \( g \), \( \sigma(f) = \{f, g\} \), \( f.bchain:=\{o\}-\{j\} \), attach cycle \( \{g\}-\{i\}-\{h\}-\{g\} \) to \( f \).

(xiii) \( \text{dfs} \) backtracks to \( c \): eject \( f \), rename cycle \( \{g\}-\{i\}-\{h\}-\{g\} \) attached to \( f \) as \( \{f\}-\{i\}-\{h\}-\{f\} \), \( behain_j: \{o\}-\{j\}-\{f\} \).

(xiv) Since \( \hat{P}(c) \preceq \hat{P}(f) \), convert \( behain_j \) into cycle \( \{c\}-\{o\}-\{j\}-\{f\}-\{c\} \) and attach to \( c \).

(xv) \( \text{dfs} \) backtracks to \( b \): \( b \) absorbs \( c \), \( \sigma(b) = \{b, c\} \), attach cycle \( \{c\}-\{o\}-\{j\}-\{f\}-\{c\} \) to \( b \).

(xvi) \( \text{dfs} \) backtracks to \( a \): ejects \( b \), rename cycle \( \{c\}-\{o\}-\{j\}-\{f\}-\{c\} \) attached to \( b \) as \( \{b\}-\{o\}-\{j\}-\{f\}-\{b\} \), \( d.tchain: \{b\} \).

(xvii) Then \( a \) absorbs \( d \), \( \sigma(a) = \{a, d\} \), convert \( d.tchain \) into cycle \( \{a\}-\{b\}-\{a\} \). The cactus representation of the cut-pair consists of cycles: \( \{a\}-\{b\}-\{a\} \), \( \{b\}-\{o\}-\{j\}-\{f\}-\{b\} \), \( \{f\}-\{i\}-\{h\}-\{f\} \), and \( \{j\}-\{\ell\}-\{j\} \).

\( \square \)