A linear-time certifying algorithm for recognizing generalized series-parallel graphs

Francis Y.L. Chin[*], Hing-Fung Ting[*], Yung H. Tsin[†], Yong Zhang[‡]

Abstract

The problems of recognizing series-parallel graphs, outerplanar graphs, and generalized series-parallel graphs have been studied separately in the past. Efficient algorithms have been presented. However, none of the algorithms are certifying. A certifying algorithm generates, in addition to its answer, a certificate that can be used by a checker (a separate algorithm) to verify the correctness of the answer. The certificate is positive if the answer is 'yes', and is negative if the answer is 'no'. In this paper, an $O(|E|+|V|)$-time certifying algorithm that simultaneously determines if a multigraph (a graph that may have parallel edges but not self-loops) $G=(V,E)$ is series-parallel, outerplanar, or generalized series-parallel is presented. The positive certificates are a construction sequence for constructing G if G is series-parallel, a generalized construction sequence for constructing G if G is generalized series-parallel but not series-parallel, and the edge set of the exterior boundary of an outerplanar embedding of G if G is outerplanar. The negative certificates are forbidden subgraphs or forbidden structures of G. All these certificates are generated by making only one pass over G after a preprocessing step decomposing G into its biconnected components.

Keywords: graph algorithm, certifying algorithm, recognition algorithm, ear-decomposition, depth-first search, series-parallel graph, outerplanar graph, generalized series-parallel graph, forbidden structure, certificate, certificate authentication.

1 Introduction

A major problem in software development is the correctness of software. Even after the designers proved the correctness of their algorithm, there is no guarantee that the algorithm will be implemented correctly as a program. This is particularly true for non-trivial algorithms as their implementation tends to be error-prone. To eliminate the bugs (implementation errors) in the program, the implementer tests their program with some test sets. Clearly, it is unlikely that they can eliminate all the bugs with this method. As a consequence, when a user gives x as an input to the program and gets output y, they usually cannot tell if y is actually a correct output or is an incorrect output caused by an undetected bug in the program.

Kratsch et al. [13] addressed this problem by introducing certifying algorithms. A *certifying algorithm* is an algorithm that, on input x, produces an output y with a certificate w that the output y is correct. By

[*]Department of Computer Science, the University of Hong Kong, Hong Kong; {chin, hfting}@cs.hku.hk
[†]School of Computer Science, University of Windsor, Windsor, Ontario, Canada, N9B 3P4; peter@uwindsor.ca
[‡]Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences; zhangyong@siat.ac.cn

checking w with an *authentication algorithm* (a program verifying that w proves that y is a correct output for x), the user is certain that y is the correct output for input x. A major merit of this approach is that even if the program is not bug-free, the user can be confident that the output they received for a particularly input has not been compromised by a bug. Certifying algorithms have been used in the library LEDA [15].

It has been observed that many graph optimization problems that are NP-hard for arbitrary graphs can be solved in polynomial time for some restricted classes of graphs which are of practical interest. Designing algorithms that are capable of recognizing such restricted classes is of theoretical and practical importance. These algorithms are called *recognition algorithms* as they return a 'yes' if the input graph is in a restricted class and a 'no' otherwise. A number of certifying algorithms for recognizing some classes of graphs have been proposed [2, 4, 6, 13, 16]. In this paper, we study the recognition of three classes of graphs: series-parallel graphs, outerplanar graphs and generalized series-parallel graphs. All of them have polynomial-time algorithms for problems, such as the Hamiltonian cycle problem and the minimum vertex-cover problem, which are NP-complete or NP-hard for general graphs [9, 18, 22].

The problem of determining if a graph is series-parallel has been studied. Linear-time algorithms were proposed [20, 26]. These algorithms are based on the following property of series-parallel graphs: a graph G is series-parallel if and only if it can be reduced to the complete graph K_2 by repeatedly applying the following two operations: (i) replace a vertex of degree two and its two incident edges with a new edge, (ii) replace two parallel edges with an edge connecting their common end-vertices. Since the algorithms just output a 'yes' or 'no', they are not certifying. Likewise, a number of linear-time algorithms have been proposed for recognizing outerplanar graphs. Brehaut [1] proposed two algorithms that both rely heavily on the planarity testing algorithm of Hopcroft et al. [10] and are thus very complicated. Sysło et al. [21] presented a simpler algorithm based on the property that a biconnected graph is outerplanar if and only if it is a cycle or it can be reduced to a cycle by contracting maximal paths to edges. Mitchell [17] presented an algorithm using the idea that a biconnected outerplanar graph can be transformed into a maximal outerplanar graph which can be recognized by removing vertices of degree 2 until only two adjacent vertices remain. Wiegers [27] presented yet another algorithm by removing vertices of degree two or one until an edgeless graph is obtained. None of the aforementioned algorithms produce an outerplanar embedding if the graph is outerplanar and none of them are certifying. It is well-known that the problem of recognizing outerplanar graphs can be reduced to that of recognizing planar graphs and the resulting algorithm can be made certifying [14]. However, as the algorithm reduces the simple outerplanar graph recognition problem to the much complicated planar graph recognition problem, and hence uses the complicated planarity testing algorithm of Hopcroft et al., it is unnecessarily complicated in comparison with ours. Besides, it is not obvious as to how to modify the algorithm so that it would determine if G is series-parallel or generalized series-parallel at the same time. Wimer and Hedetniemi [29] outlined a recognition algorithm for generalized series-parallel graphs. Their algorithm is non-certifying and does not distinguish generalized series-parallel graphs that are also series-parallel from those that are not.

Let $\mathtt{SP}$, $\mathtt{OP}$, and $\mathtt{GSP}$ be the class of series-parallel graphs, outerplanar graphs, and generalized series-parallel graphs, respectively. It is known that $\mathtt{OP}, \mathtt{SP} \subsetneq \mathtt{GSP}$, $\mathtt{OP} \nsubseteq \mathtt{SP}, \mathtt{SP} \nsubseteq \mathtt{OP}$, and $\mathtt{SP} \cap \mathtt{OP} \neq \emptyset$. Hence, $\mathtt{GSP}$ can be partitioned into four subclasses (Figure 1). In this paper, we present the first certifying recognition algorithm that determines if a multigraph $G = (V, E)$ is $\mathtt{GSP}$ and if it is, to which subclass it belongs in $O(|V| + |E|)$ time. For instance, if $G \in \mathtt{OP} \setminus \mathtt{SP}$, then two positive certificates are generated for its membership in $\mathtt{GSP}$ and $\mathtt{OP}$ and a negative certificate is generated for its non-membership in $\mathtt{SP}$.

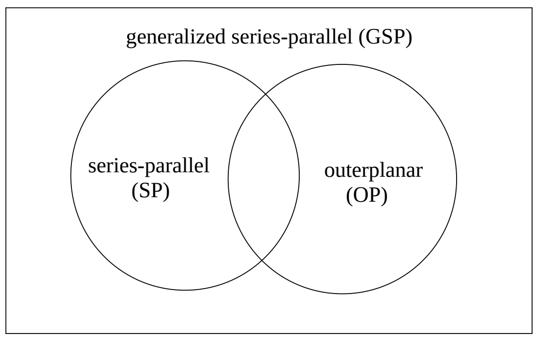

Figure 1: The classes $\mathtt{SP}$, $\mathtt{OP}$ and $\mathtt{GSP}$.

Our algorithm also differs from the existing non-certifying algorithms in the following ways: firstly, the existing algorithms either use graph contraction techniques to reduce the given graph to a single edge, a cycle, or an edgeless graph, or reduce the problem to the planar graph problem. It is not obvious as to how to modify them to turn them into certifying algorithms. Our algorithm uses depth-first search to decompose the given graph into a collection of paths based on which the desired certificates are generated. Secondly, as is shown in Figure 1, the three classes of graphs are closely related. Therefore, the algorithms for solving them should form a cohesive and succinct unit like ours. This is not the case for existing algorithms as they were designed independently. Hence, while our algorithm makes only one pass over the graph, three passes are required if existing algorithms are used. Thirdly, our depth-first-search-base path decomposition technique might provide a basis for solving other graph-theoretic problems. Tsin [25] has recently used this technique to develop a certifying algorithm for the 3-edge-connectivity problem. The algorithm shares a characteristic of our algorithm in that it generates the 3-edge-connected components, a certificate for each of them, a cactus representation of the cut-pairs if the graph is not 3-edge-connected, and all the bridges if the graph is not 2-edge-connected seamlessly by making only one pass over the input graph.

This paper is organized as follows: Section 2 gives the definitions. Section 3 presents depth-first-search-based characterization theorems for biconnected series-parallel graphs and outerplanar graphs. Section 4 presents a certifying algorithm for recognizing biconnected series-parallel graphs and outerplanar graphs. Section 5 presents the authentication algorithms. Section 6 generalizes Section 4 to handle non-biconnected graphs. Section 7 presents a certifying algorithm for recognizing generalized series-parallel graphs, SP graphs and outerplanar graphs, simultaneously.

2 Definitions

An undirected graph is represented by $G = (V, E)$, where V is the vertex set and E is the edge set. An edge with end-vertices u and v is presented by (u, v) or (v, u). G is a *simple* graph if it contains no *parallel edges* (edges sharing the same end-vertices) nor *self-loop*s (edges whose end-vertices are identical). G is a *multigraph* if it may contain parallel edges but not self-loops. The *degree* of vertex w in G, denoted by $deg_G(w)$, is the number of edges having w as an end-vertex.

A sequence of vertices $v_0v_1 \dots v_k$ is a *path* if $(v_i, v_{i+1}) \in E, 0 \leq i < k$, and $v_i, 0 \leq i < k$, are distinct except v_k which may be identical to v_0. The path is a *cycle* if $v_k = v_0$ and $k \geq 2$. The path is a *null path* if $k = 0$. The path is also called a $v_0 - v_k$ *path* and vertices v_0 and v_k are its *terminating vertices* while $v_i, 1 \leq i \leq k-1$, are its *internal vertices*. A graph is *connected* if for every two vertices u and v, there is an $u - v$ path. A graph is a *tree* if it is connected and has no cycle. A *cut-vertex* in a connected graph is a vertex whose removal results in a disconnected graph. A connected graph is *biconnected* if it has no cut-vertex. A pair of vertices is a *separation pair* of a connected graph if their removal results in a disconnected graph and neither is a cut-vertex. A graph $G' = (V', E')$ is a *subgraph* of $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. If G' is a tree and $V' = V$, then it is a *spanning tree* of G. A *biconnected component* of a graph is a maximal biconnected subgraph.

Traversing a graph $G = (V, E)$ with a depth-first search [23] (henceforth abbreviated as dfs) creates a spanning tree $T = (V, E_T)$, called the *depth-first search tree* (abbr. dfs tree) of G. T is a rooted tree rooted at vertex r where the search begins. Every vertex u is assigned a distinct integer, $dfs(u)$, called its *dfs number*, which is its rank in the order the vertices are visited by the search for the first time. An edge of G is a *tree-edge* if it belongs to T and is a *back-edge* otherwise. For all $w \in V \setminus \{r\}$, there is a unique tree-edge (u, w) such that $dfs(u) < dfs(w)$. Vertex u is called the *parent* of w, denoted by $parent(w)$, while w is a *child* of u. Furthermore, the edge is the *parent edge* of w and a *child edge* of u. Since (u, w) is the only parent edge of w, it can be uniquely represented by $(parent(w) \rightarrow w)$. A *leaf* is a vertex with no child. A *tree-path* is a path in T. Vertex u is an *ancestor* of vertex v, denoted by $u \preceq v$, if and only if u lies on the $r - v$ tree-path. Vertex u is a *proper ancestor* of v, denoted by $u \prec v$, if and only if $u \preceq v$ and $u \neq v$. Vertex v is a (*proper*) *descendant* of u if and only if u is a (proper) ancestor of v. Note that if $u \prec v$, then $dfs(u) < dfs(v)$. Every back-edge connects an ancestor with a descendant. A back-edge (u, v) is an *outgoing back-edge* (*incoming back-edge*, resp.) of u (v, resp.) if $v \prec u$. The *height* of a vertex v in T is: $height(v) = 0$ if v is a leaf; $height(v) = \max\{height(u) \mid u \text{ is a child of } v\} + 1$, otherwise. The *subtree* of T rooted at vertex w, denoted by T_w, is the subgraph of T induced by the set of descendants of w. V_{T_w} denotes the vertex set of T_w. An *embedding* of a graph is a graphical representation of the graph on the plane. A *planar embedding* is an embedding in which no two edges intersect except possibly at their end-vertices. A *face* of a planar embedding is a maximal region of the plane that is bounded by some edges of the graph and contains no edges within it; the edges form the *boundary* of that face. The *exterior face* is the face that has unbound area. The *exterior boundary* is the boundary of the exterior face. An *outerplanar embedding* is a planar embedding in which all the vertices lie on the exterior boundary. An *edge-subdivision* is an operation that replaces an edge with a path of length two whose internal vertex is a new vertex. A *subdivision* of a graph G is a graph that can be obtained from G by a sequence of edge-subdivisions. The graph $K_{2,3}$ is the complete bipartite graph whose bipartition contains two vertices in one set and three vertices in the other set; the graph K_4 is the complete graph with four vertices; (Figure 2).

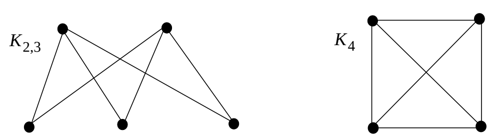

Figure 2: The graphs $K_{2,3}$ and K_4.

In the sequel, an edge $(u, v) \in E$ is denoted by $(u \rightarrow v)$ if it is a tree-edge with u as the parent, or by $(v \curvearrowleft u)$ if it is an outgoing back-edge of u. Moreover, $s(v \curvearrowleft u) = u$ and $t(v \curvearrowleft u) = v$. A path with u and v as terminating vertices and with an orientation from u to v is denoted by $u \rightsquigarrow v$ with $s(u \rightsquigarrow v) = u$;

$t(u \rightsquigarrow v) = v$. If the path is a tree-path, it is denoted by $u \rightsquigarrow_T v$. If the path is a section of another path P, it is also denoted by $u \rightsquigarrow_P v$.

An undirected multigraph is a *generalized series-parallel* (abbr. GSP) *graph* with source s and sink t if it can be constructed recursively as follows:

- Every edge $e = (u, v)$ is a GSP graph with u designated as the source and v designated as the sink.
- Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two disjoint GSP graphs with source s_1, sink t_1, and source s_2, sink t_2, respectively. A new GSP graph is created from G_1 and G_2 by
  - the *series composition* **SC**(G_1, G_2): identify t_1 with s_2 and designate s_1 and t_2 as its source and sink respectively, or
  - the *parallel composition* **PC**(G_1, G_2): identify s_1 with s_2 and t_1 with t_2, and designate s_1 and t_1 as its source and sink, respectively.
  - the *dangling composition* **DC**(G_1, G_2): identify s_1 with s_2, and designate s_1 and t_1 as its source and sink, respectively.

Removing the DC operation and replacing every occurrence of GSP with SP in the above definition, we have the definition of *series-parallel* (abbr. SP) graph. Clearly, SP graphs are GSP graphs but not vice versa. The sequence of composition operations used to construct a GSP (SP, respectively) graph is called a *construction sequence* of the graph.

An *outerplanar graph* is a graph that has an outerplanar embedding.

3 Characterization theorems

First, we shall consider how to recognize SP graphs and outerplanar graphs that are biconnected. Our algorithm is based on open ear-decomposition of undirected graphs generated by depth-first search and the following theorems that state forbidden subgraphs of SP graphs and outerplanar graphs.

Theorem 3.1. [3] *A biconnected graph G is SP if and only if it does not contain a subdivision of K_4.*

Theorem 3.2. [8] *A biconnected graph G is outerplanar if and only if it does not contain a subdivision of $K_{2,3}$ or K_4.*

Corollary 3.2.1. *Every biconnected outerplanar graph is SP.*

Definition: A *ear-decomposition* of a connected graph $G = (V, E)$ is a partition of E into a sequence of edge-disjoint paths $P_i, 1 \leq i \leq k$, such that for every $P_i, 2 \leq i \leq k$, each terminating vertex of P_i lies on an $P_j (j < i)$ and no internal vertex of P_i lies on any $P_j (j < i)$. Each P_i is called an *ear*. $P_i, 1 \leq i \leq k$, is an *open-ear decomposition* if $k = 1$ and P_1 is an edge, or $k > 1$, P_1 is a cycle and $P_i, 2 \leq i \leq k$, is a path with distinct terminating vertices.

Lemma 3.3. [28] *G is biconnected if and only if it has an open-ear decomposition.*

Ear decompositions have been used to characterize several graph connectivity properties. Based on these characterizations, $O(\lg n)$-time parallel algorithms for recognizing the graph connectivity properties on the PRAM have been developed [7, 11]. Eppstein showed that a biconnected graph is SP if and only if it has a *nested* ear decomposition and based on this characterization he designed an $O(\lg n)$-time parallel algorithm for the PRAM [5]. In the following, we give SP graphs a new characterization based on ear-decomposition and depth-first search and then present a linear-time algorithm based on it. Since depth-first search is inherently sequential [19] and our algorithm uses the sequential date structure *stack* heavily, our algorithm and Eppstein's algorithm use completely different approaches. Moreover, Eppstein's algorithm is not certifying and does not recognize outerplanar graphs at the same time.

Let $G = (V, E)$ be a biconnected simple graph with $|E| \geq 2$. By performing a depth-first search over G, we can use the dfs numbers of the vertices to rank the back-edges as follows [24].

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

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

(ii) $dfs(q) = dfs(y)$ and $dfs(p) < dfs(x)$ and $p \not\prec x$, or

(iii) $dfs(q) = dfs(y)$ and $x \prec p$.

Using the back-edges and their ranks in lexicographical order, the edges of G can be partitioned into a collection of edge-disjoint paths such that every path contains exactly one back-edge as follows: first recall that every tree-edge can be uniquely represented by $(parent(u) \rightarrow u)$ for some $u \in V$. For each tree-edge $(parent(u) \rightarrow u)$, we associate with it the lexicographically smallest back-edge $(y \curvearrowleft x)$ such that $y \prec u \preceq x$. The back-edge exists because G is biconnected and $|E| \geq 2$. It is easily verified that $(y \curvearrowleft x)$ and all the tree-edges it is associated with form a path $yxw_1w_2 \dots w_kv$ in G such that $vw_k \dots w_2w_1x$ is $v \rightsquigarrow_T x$. Furthermore, if $(y \curvearrowleft x)$ has the rank i lexicographically, we denote the path by $P_i : yxw_1w_2 \dots w_kv$ and let $s(P_i) = y$ and $t(P_i) = v$. Hence the paths can also be ranked lexicographically. We also use $P_{(y \curvearrowleft x)}$ to denote P_i. It is easily verified that the sequence of paths $P_i, 1 \leq i \leq |E| - |V| + 1$, is an open ear-decomposition of G. P_i is a *non-trivial ear* if it contains at least one tree-edge and is a *trivial ear* otherwise. Notice that for each back edge $(v \curvearrowleft u)$, $s(v \curvearrowleft u) = u$ and $t(v \curvearrowleft u) = v$, but when it is treated as a trivial ear P, $s(P) = v$ and $t(P) = u$.

It is important to point out that the ear-decomposition is not generated explicitly. 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 $ear(e)$, is determined during the depth-first search based on the following recursive definition:

$$ear(e) = \begin{cases} e & \text{if } e \in E \backslash E_T; \\ \min_{\lessdot}(\{f \mid f = (v \curvearrowleft w) \in E \backslash E_T\} \cup & \\ \qquad \{ear(f) \mid f = (w \rightarrow v) \in E_T\}), & \text{if } e = (parent(w) \rightarrow w) \in E_T \end{cases}$$

For each vertex $w (\neq r)$, Let $ear(parent(w) \rightarrow w) = f'$. Then of all the ears that contain either a child edge or an outgoing back edge of w, $P_{f'}$ is the only ear that can be extended to include the parent edge of w. The remaining ears all terminate at w.

Definition: A vertex v *strongly belongs* (or *s-belongs*) to P, denoted by $v \in_s P$, where P is an ear or a section of an ear if the parent edge of v is an edge on P [16]. An ear P_i is *strongly attached* (or *s-attach*) to P if $t(P_i) \in_s P$ and $s(P_i)$ belongs to P. An ear P_i is s^**-attached* to P if P_i is s-attached to P or P_i is s-attached to an ear that is s^*-attached to P. Two ears P_h and P_k are *interlacing* if they both s-attached to a ear P_i such that $s(P_h) \prec s(P_k) \prec t(P_h) \prec t(P_k)$.

The following is a characterization theorem for SP graphs that is based on an open ear-decomposition generated by a depth-first search and Theorem 3.1.

Theorem 3.4. *Let $P_1, P_2, \cdots, P_{|E|-|V|+1}$ be the ears of a biconnected simple graph $G = (V, E)$ generated by a depth-first search in lexicographical order. Then G is SP if and only if the following conditions hold:*

(a) *For every ear $P_i, i > 1$, there exists an ear $P_j (j < i)$ to which P_i is s-attached;*

(b) *For every ear P_i, there do not exist two interlacing ears that are both s-attached to P_i.*

Proof. Let G be an SP graph.

(a) Suppose to the contrary that there exists an $P_i(i > 1)$ not s-attached to any ear $P_h(h < i)$ (Figure 3(a)). Let $t(P_i) \in_s P_j$. Then $s(P_i)$ does not belong to P_j and $P_j \lessdot P_i$ imply that $s(P_j) \prec s(P_i) \prec t(P_j)$. Moreover, $t(P_i) \in_s P_j$ implies $t(P_j) \prec t(P_i)$. We thus have $s(P_j) \prec s(P_i) \prec t(P_j) \prec t(P_i)$. Since $s(P_j) \prec t(P_j)$, $P_j \neq P_1$ which implies that $t(P_j) \in_s P_k$, for some $k < j$. Then $P_k \lessdot P_j$ which implies that $s(P_k) \preceq s(P_j)$. Clearly, $s(P_j) \rightsquigarrow_T t(P_j)$ and P_j form a circle which with P_i, P_k and $s(P_k) \rightsquigarrow_T s(P_j)$ form a K_4-subdivision, contradicting Theorem 3.1.

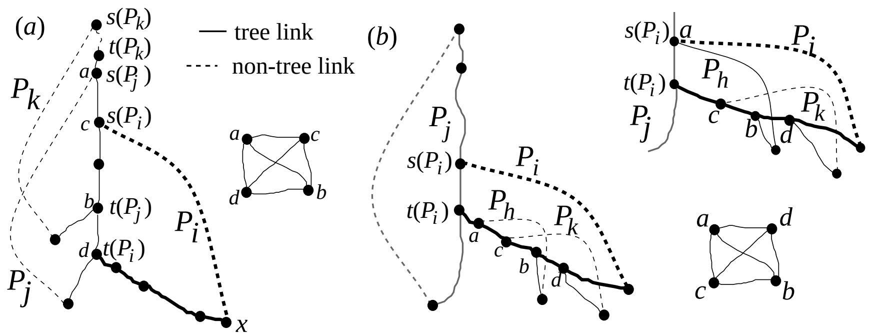

Figure 3: Forbidden structure K_4 minor.

(b) Suppose to the contrary that for some P_i, there exist two interlacing ears P_h and P_k s-attached to P_i. (Figure 3(b)) Then, P_i, P_h, P_k and $s(P_i) \rightsquigarrow_T t(P_i)$ form a subdivision of K_4, contradicting Theorem 3.1.

Conversely, suppose Conditions (a) and (b) hold for $G = (V, E)$. Let $G_i, 1 \leq i \leq |E| - |V| + 1$, be the graph consisting of $P_1, P_2, \cdots, P_i$. We shall apply induction on i to prove that each G_i is SP.

G_1 is a cycle which is obviously SP. Suppose the assertion holds for $i < m(\geq 2)$. Consider adding P_m to G_{m-1}. By the induction hypothesis, G_{m-1} is SP. By Conditions (a), P_m is s-attached to an ear $P_j, j < m$. If $s(P_m) = s(P_j)$, then there is no ear P_i of G_{m-1}, hence of G, such that $(s(P_m) =)s(P_j) \prec s(P_i) \prec t(P_m) \prec t(P_i)$ or P_m and P_i would be interlacing ears, contradicting Condition (b). But then P_m can be merged into P_j, first with a PC operation merging P_m with the SP subgraph consisting of $s(P_m) \rightsquigarrow_{P_j} t(P_m)$ and all the ears s*-attached to it, then with an SC operation joining the resulting SP-subgraph with the SP-subgraph consisting of $t(P_m) \rightsquigarrow_{P_j} t(P_j)$ and all the ears s*-attached to it. The SP graph G_m is then formed. If $s(P_m) \neq s(P_i)$, By Conditions (b), there is no ear P_i of G_{m-1}, hence of G, such that $t(P_i)$ ($s(P_i)$, resp) is an internal vertex of the tree-path $s(P_m) \rightsquigarrow_T t(P_m)$ while $s(P_i)$ ($t(P_i)$, resp) lies outside the tree-path. Hence, $\{s(P_m), t(P_m)\}$ is a separation pair partitioning G_{m-1} into two or more connected components each of which is an SP graph with $t(P_m)$ and $s(P_m)$ as the source or sink. Since

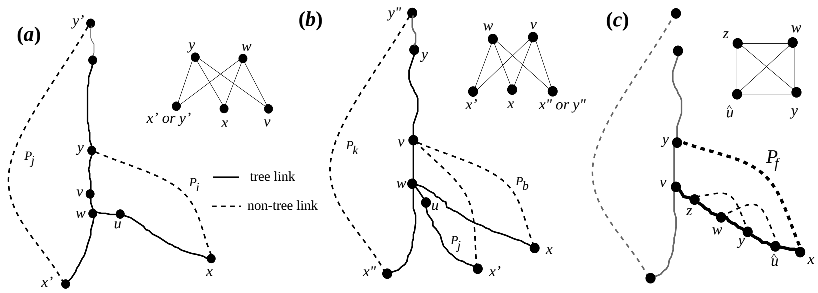

Figure 4: A characterization of outerplanar graph.

P_m is an SP graph with source $t(P_m)$ and sink $s(P_m)$, it can be merged with those SP graphs with source $t(P_m)$ and sink $s(P_m)$ to form G_m. $\square$

For outerplanar graphs, we have the following characterization theorem whose correctness is based on an open ear-decomposition generated by a depth-first search and Theorem 3.2 [24].

Theorem 3.5. *Let $P_1, P_2, \cdots, P_{|E|-|V|+1}$ be the ears of a biconnected simple graph $G = (V, E)$ generated by a depth-first search in lexicographical order. Then G is not outerplanar if and only if one of the following conditions holds (Figure 4):*

(a) *There exists a non-trivial ear $P_i, i \geq 2$ such that $s(P_i) \neq parent(t(P_i))$, or*

(b) *$\exists e \in E_T$ for which there are two non-trivial ears P_i, P_j such that $e = (s(P_i) \to t(P_i)) = (s(P_j) \to t(P_j))$, or*

(c) *There is an ear $P_i, i \geq 1$, to which two interlacing trivial ears are s-attached.*

Condition (a) ((b), respectively) implies that G contains a $K_{2,3}$-subdivision (Figures 4(a),(b)) while Condition (c) implies that G contains a K_4-subdivision (Figure 4(c)). By Theorem 3.2, G is not outerplanar.

4 Recognizing SP graphs and outerplanar graphs

Let $G = (V, E)$ be a biconnected multigraph graph. The *underlying simple graph* of G is the simple graph $G_s = (V, E_s)$ such that $(u, v)^\ell \in E_s$ if and only if vertices u and v are connected by ℓ parallel (u, v) edges in G. Specifically, every set of ℓ parallel (u, v) edges in G is replaced by a single edge $(u, v)^\ell$ in G_s. The graph G_s can be represented by the following compact adjacency-lists structure:

- for each $(u, v)^\ell \in E_s$, there exists $|\underline{\overline{\ell\|v}}|$ ($|\underline{\overline{\ell\|u}}|$, respectively) in the adjacency list $L[u]$ ($L[v]$, respectively) of u (v, respectively); $|\underline{\overline{\ell\|u}}|$ has a pointer pointing at $|\underline{\overline{\ell\|u}}|$, and vice versa.

This compact adjacency-lists structure can be constructed in $O(|E|)$ time by sorting E in lexicographical order with radix sort following by a scan over the sorted list. It is easily verified that G is SP (outerplanar, respectively) if and only if G_s is SP (outerplanar, respectively). Hence, the problem of recognizing SP and outerplanar multigraphs can be reduced to that of recognizing SP and outerplanar simple graphs. Theorems 3.4 and 3.5, can thus be applied. Dealing with G_s instead of G not only simplifies the presentation of our algorithms as we do not need to deal with parallel edges but also reduces the number of $\texttt{PC}$ operations performed, hence the size of the data structure representing the construction sequence. Since G_s is a simple graph, if G_s is SP or outerplanar, then $|E_s| \leq 2|V| - 3$ [12]. Hence, the size of $L[v], v \in V$, is bounded by $O(|V|)$. This implies that using the compact adjacency lists, the recognition algorithms run in $O(|V|)$ time.

Our algorithm performs a depth-first search over G_s attempting to construct a construction sequence and an exterior boundary of G_s. When the search backtracks to the root r, if G_s is in $\texttt{SP} \cap \texttt{OP}$, a construction sequence and an exterior boundary of G_s are generated; if G_s is in $\texttt{SP} \setminus \texttt{OP}$, a construction sequence of G_s and a $K_{2,3}$-subdivision of G_s are generated. If G_s is not in $\texttt{SP}$, execution of the algorithm is aborted and a K_4-subdivision of G_s is generated. Note that for biconnected graphs, $\texttt{OP} \setminus \texttt{SP} = \emptyset$ by Corollary 3.2.1. For clarify, we shall address how to recognize SP graphs and outerplanar graphs separately.

Since the parallel-edge counts ℓ are kept in the nodes of the adjacency lists, the compact adjacency-lists structure also represents G. Therefore, in the following discussion, we shall use G and G_s interchangeably.

4.1 Recognizing series-parallel graphs

A construction sequence of an SP graph G can be conveniently represented by a binary tree $\mathcal{T}_G$, called a *decomposition tree* (Figure 4), similar to that of minimal vertex series-parallel graph [26] as follows:

- T_G consists of a single node $|\overline{\underline{\ell\|u|\mathbf{e}|v}}|$, if G is a set of ℓ parallel edges with source u and sink v.
- T_G is a binary tree with $|\overline{\underline{0\|s|\mathbf{S}|t}}|$ as the root, T_{G_1} and T_{G_2} as the left and right subtrees, respectively, if $G = \texttt{SC}(G_1, G_2)$, where s is the source of G_1 and t is the sink of G_2.
- T_G is a binary tree with $|\overline{\underline{0\|s|\mathbf{P}|t}}|$ as the root, T_{G_1} and T_{G_2} as the left and right subtrees, respectively, if $G = \texttt{PC}(G_1, G_2)$, where s and t are the common source and sink of G_1 and G_2, respectively.

Note that by replacing every $|\overline{\underline{\ell\|u|\mathbf{e}|v}}|$ node with a binary tree consisting of ℓ leaf nodes $|\overline{\underline{u|\mathbf{e}|v}}|$ and $\ell - 1$ internal nodes $|\overline{\underline{u|\mathbf{P}|v}}|$; every $|\overline{\underline{0\|s|\mathbf{S}|t}}|$ with $|\overline{\underline{s|\mathbf{S}|t}}|$, and every $|\overline{\underline{0\|s|\mathbf{P}|t}}|$ with $|\overline{\underline{s|\mathbf{P}|t}}|$, we can turn $\mathcal{T}_G$ into a conventional decomposition tree in $O(|E|)$ time.

In explaining how to generate a decomposition tree of G in detail, the following notations will be used:

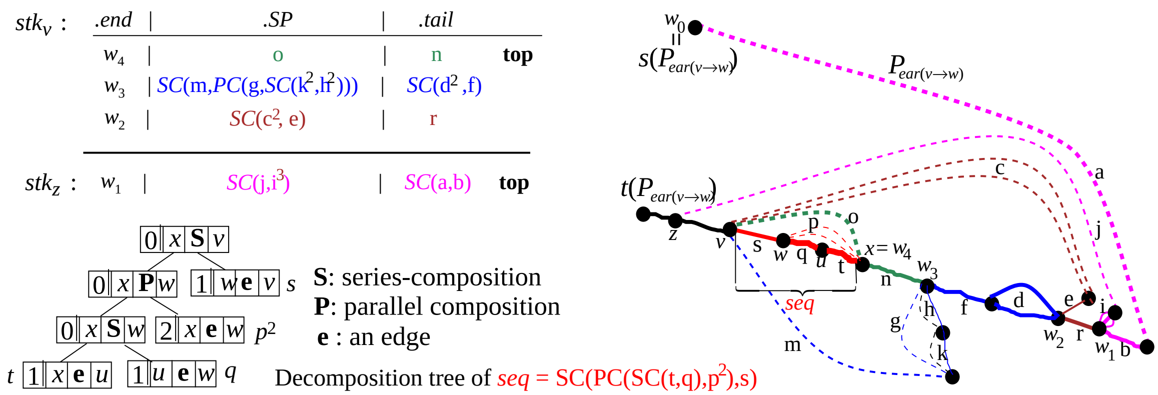

Figure 5: *dfs* backtracks from w to v, the w-SPchain, stacks stk_v, stk_z and a decomposition tree of seq.

$SP_{x \rightsquigarrow y}$**:** an SP subgraph consisting of the path $x \rightsquigarrow y$ (which is a section of an ear) and all the ears s*-attached to it. The source and sink of $SP_{x \rightsquigarrow y}$ are x and y, respectively.

$SP_{x,y}$**:** an SP subgraph consisting of an ear P and all the ears s*-attached to P, such that $x = t(P)$ is the source and $y = s(P)$ is the sink unless $y = r$; then $r = s(P)$ is the source and $x = t(P)$ is the sink.

Both $SP_{x \rightsquigarrow y}$ and $SP_{x,y}$ are represented by decomposition tree.

At each vertex w, a stack stk_w is maintained. An entry **x** on stk_w has three fields: **x**.SP, **x**.end and **x**.$tail$, where **x**.$SP = SP_{x,w}$, for some x, **x**.$end = x$, and **x**.$tail = SP_{z \rightsquigarrow x}$, for some z, or nil (Figure 5). If G is SP, entry **x**$'$ is above entry **x**$''$ on stk_u if and only if $x' \prec x''$. When the dfs backtracks to w, all the SP subgraphs stored on stk_w are popped and merged to form a larger SP subgraph. The top entry is represented by **top**.

The key idea of the algorithm is to perform a depth-first search over G so that when the dfs backtracks from a vertex w to its parent v, the parent edge $(w, v)^p$ and the section of ear $P_{ear(v \rightarrow w)}$ from $s(P_{ear(v \rightarrow w)})$ to w and all the ears s*-attached to that section have been transformed into a chain of SP subgraphs, called the w-SPchain, $SP_{w_i \rightsquigarrow w_{i+1}}, 0 \leq i < k$, and $SP_{w_k \rightsquigarrow v}$, where $w_0 = s(P_{ear(v \rightarrow w)})$, such that (Figure 5):
(i) For each ear P_f that is not s-attached to the aforementioned section of $P_{ear(v \rightarrow w)}$ but $t(P_f)$ s-belongs to that section or $t(P_f) = w$ (note that $s(P_f) \prec w$), the ear and all the ears s*-attached to it have been transformed into an $SP_{w_i,s(P_f)}$, for some $1 \leq i \leq k$, and stored as **x**.SP in some entry **x** on stack $stk_{s(P_f)}$; (ii) for each $w_i, 1 \leq i \leq k$, there is at least one $SP_{w_i,s(P_f)}$; (iii) every $SP_{w_{i-1} \rightsquigarrow w_i}, 1 \leq i \leq k$, is stored on $stk_{\tilde{s}_{w_i}}$ as **top**.$tail$, where $\tilde{s}_{w_i} = s(P_{\tilde{f}})$ and $P_{\tilde{f}}$ is the lexicographically smallest ear with $t(P_{\tilde{f}}) = w_i$. The only SP subgraph that is not stored on any stack is $SP_{w_k \rightsquigarrow v}$ which is designated as seq_w or seq. In Figure 5, the w-SPchain consists of $SP_{w_0 \rightsquigarrow w_1}$ = SC(a,b), $SP_{w_1 \rightsquigarrow w_2} =$ r, $SP_{w_2 \rightsquigarrow w_3}$ = SC(d^2,f), $SP_{w_3 \rightsquigarrow w_4}$ = n, and

$seq = SP_{w_4 \rightsquigarrow v} =$ SC(PC(SC(t,q),p^2),s). The w-SPchain is constructed as follows:

When w is a leaf of the dfs tree, (a) if w has no outgoing back edge, then as G is biconnected, $v = r$ and G consists of the set of parallel edges $(w, v)^p$, where $p \geq 1$. Hence $(w, v)^p$ represents a construction sequence of G and execution of the algorithm terminates. (b) if w has exactly one outgoing back edge $(u \curvearrowleft w)^\ell$, the w-SPchain consists of $SP_{u \rightsquigarrow v} (= seq_w) = \mathsf{SC}((u \curvearrowleft w)^\ell, (w, v)^p)$. In this case, $k = 0$. (c) If w has more than one outgoing back edges, the w-SPchain consists of $SP_{\tilde{u} \rightsquigarrow w}$ and $SP_{w \rightsquigarrow v}$, where $SP_{\tilde{u} \rightsquigarrow w} = (\tilde{u} \curvearrowleft w)^\ell$ which is the lexicographically *smallest* outgoing back edge of w and $SP_{w \rightsquigarrow v} = (w, v)^p$. Each remaining outgoing back edge $(u' \curvearrowleft w)^h$ contributes one $SP_{w,u'}$ which is stored on stack $stk_{u'}$. Moreover, $(\tilde{u} \curvearrowleft w)^\ell$ is stored as **top**.$tail$ on stack $stk_{\tilde{s}_w}$. In this case, $k = 1$, $w_1 = w$ and $seq_w = (w, v)^p$.

When w is an internal vertex, when the dfs backtracks to w from a child vertex u, the u-SPchain has been constructed. Stack stk_w, if non-empty, is popped to extend seq_u. If there is an ear interlacing with some ears stored on stk_w, it will be detected and a K_4-subdivision is generated (Figure 6(a)). Otherwise, when stk_w is emptied, (a) If $t(ear(w \to u)) \succ t(ear(v \to w))$, the u-SPchain must consists of solely seq_u which is pushed onto stack $stk_{t(ear(w \to u))}$ or a K_4-subdivision is returned (Figure 6(b)). (b) If $t(ear(w \to u)) \prec t(ear(v \to w))$, then as with Case (a), the current w-SPchain must consists of solely seq_w which is pushed onto stack $stk_{t(ear(v \to w))}$ or a K_4-subdivision is returned. The u-SPchain then becomes the current w-SPchain, $seq_w := seq_u$ and $ear(v \to w) := ear(w \to u)$. (c) If $t(ear(w \to u)) = t(ear(v \to w))$, then as with cases (a) and (b), the u-SPchain and the current w-SPchain are seq_u and seq_w, respectively, or a K_4-subdivision is returned. The two SPchains are merged to form the current w-SPchain consisting of solely seq_w which is the SP subgraph: $SP_{t(ear(v \to w)) \rightsquigarrow w} = \mathsf{PC}(seq_w, seq_u)$.

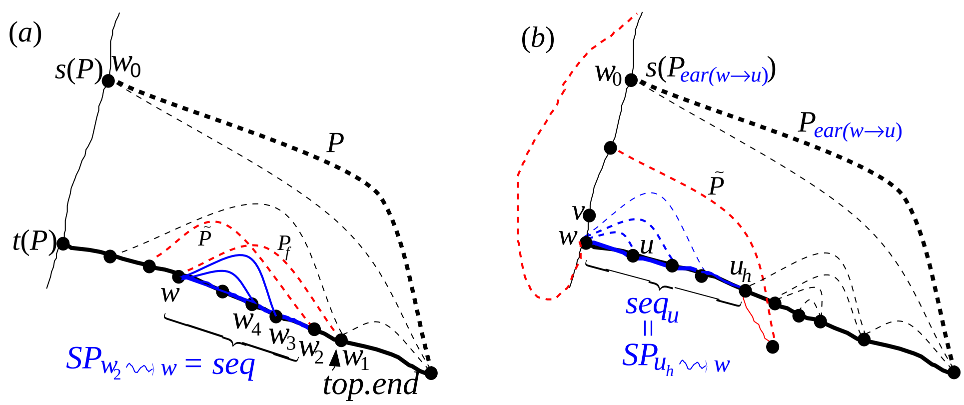

Figure 6: Detecting violation of Condition (a) and (b).

For each outgoing back edge $(u \curvearrowleft w)^h$ of w, since the edge can be viewed as an SP chain consisting of just $(u \curvearrowleft w)^h$, by letting $seq_u = (u \curvearrowleft w)^h$, the above procedure applies.

When $L[w]$ is completely processed, (a) if $v \neq r$, then if there is no ear terminating at w, $seq_w := \texttt{SC}(seq_w, (w,v)^p)$ (i.e. extend seq_w to include the parent edge); otherwise, seq_w is stored as **top**.$tail$ on stack $stk_{\tilde{s}_w}$, and $seq_w := (w,v)^p$. In either case, the dfs backtracks to v. (b) If $v = r$, then seq_w must be the entire w-SPchain which is an $SP_{r \rightsquigarrow w}$. Hence, the instruction $seq_w = \texttt{PC}((v,w)^p, seq_w)$ produces seq_w as a construction sequence of G.

The pseudo-code of the algorithm is given below. The main program and the initialization steps of Procedure `GenCS` are self-explanatory. The **for** loop in Procedure `GenCS` processes the adjacency list $L[w]$. The **if** part of the **if** statement in the loop deals with the child vertices of w while the **else** part deals with the outgoing back edges of w. Procedure `Update-seq` pops stack stk_w to update seq_u or generates a K_4-subdivision if a pair of interlacing ears is discovered. Procedure `Update-ear-of-parent` determines whether seq_w and seq_u are to be merged or one of them is to be pushed on a stack, and generates a K_4-subdivision if the one to be pushed is not the entire u-SPchain or current w-SPchain. The **if** statement following the **for** loop updates **top**.$tail$ on stack $stk_{\tilde{s}_w}$ if needed. The next **if** statement completes the construction of the construction sequence if $v = r$, or extend seq_w to include $(w,v)^p$, otherwise. Note that those instructions marked by • can be ignored for the time being as they are meant for recognizing outerplanar graphs which will be explained later.

Algorithm `SP&Outerplanar`

Input: The compact adjacency lists $L[w], \forall w \in V$, of a biconnected multigraph $G = (V,E)$.

Output: $\left\{ \begin{array}{l} seq \text{ (a } SP \text{ decomposition tree of } G) \\ E_B \text{ (the edge set of the exterior boundary of an outerplanar embedding of } G) \end{array} \right.$, if G is series-parallel and outerplanar,
or, seq (a SP decomposition tree of G) and a $K_{2,3}$ subdivision of G, if G is series-parallel but not outerplanar,
or, a K_4-subdivision of G, if G is neither series-parallel nor outerplanar;

begin
  for each $w \in V$ **do** $dfs(w) := 0$; // mark w as unvisited
    empty stk_w;
  $count := 1$; // dfs number
• $K_{2,3}$-$found := false$; // $K_{2,3}$-$found$ is $true$ iff a $K_{2,3}$-subdivision has been found
  `GenCS`$(r, \perp, 0, seq)$; // $\perp$ represents the undefined $parent(r)$;
end.

Procedure `GenCS`(w, v, p, seq) // $(w,v)^p \in E_s$
begin
  $dfs(w) := count$; $count := count + 1$; // assign a dfs number to w
  $parent(w) := v$;
  if $(w \neq r)$ **then** $ear((v \to w)) := \infty_{lex}$; // initialize $ear(v \to w)$; $f \lessdot \infty_{lex}, \forall f \in E \backslash E_T$
    $\tilde{s}_w := \infty_{\preceq}$; // initialize $\tilde{s}_w$; $u \prec \infty_{\preceq}, \forall u \in V$
• $b.alert(w) := false$; // for detecting violation of Theorem 3.5(b)
    $seq := nil$;

```
  for each (|ℓ||u| in L[w]) do            // process the adjacency list of w
    if (dfs(u) = 0) then                  // u is unvisited
        GenCS(u, w, ℓ, seq_u);
        Update-seq(w, seq_u);       // pop stk_w to update seq
        if (w ≠ r) then Update-ear-of-parent(w → u, seq_u, w, v, seq);

    else if (dfs(u) < dfs(w)) ∧ (u ≠ v) then   // outgoing back-edge (u ↶ w)^ℓ
            ear(u ↶ w) := (u ↶ w);
            Update-ear-of-parent(u ↶ w, (u ↶ w)^ℓ, w, v, seq);
  if (w ≠ r) then                                        // extend seq to include (v → w)
    if (s̃_w ≠ ∞_⪯) then for stk_{s̃_w} do top.tail := seq; seq := nil;   // there is an ear terminating at w
▶   if (v = r) then seq := PC((v, w)^p, seq)              // generate the last PC operation
              else seq := SC(seq, (w, v)^p);              // extend seq to include the parent edge of w

•  if (∼ (K_{2,3}-found)) then   // extend the exterior boundary; ∼ is the logical negation operator
•     if (w ≠ r) then
•        case (number of children of w) is
•        0: add (ear(v → w)) and (v → w) to E_B;                  // Figure 8, Case 1
•        1: if (s(ear(v → w)) = w) then add (ear(v → w)) to E_B;  // Figure 8, Case 2(a)
•                                   else add (v → w) to E_B;      // Figure 8, Case 2(b)
end; // of GenCS

Procedure Update-seq(w, seq)
begin // extend seq by merging all SP subgraphs stored on stk_w with seq
      while (stk_w is non-empty) do
          pop top from stack stk_w;
          if (source of seq ≠ top.end) then Report(K_4); stop;     // Return a K_4-subdivision
          seq := PC(seq, top.SP);  if (top.tail ≠ nil) then seq := SC(top.tail, seq);
end; // of Update-seq

Procedure Update-ear-of-parent(f, seq_u, w, v, seq)
begin
  if (t(ear(f)) ≺ t(ear(v → w))) then                              // Case (b)
          if (ear(v → w) ≠ ∞_lex) then                              // ear(v → w) is defined
•             if (∼ (K_{2,3}-found) ∧ s(ear(v → w)) ≠ w) then        // P_{ear(v→w)} is non-trivial
•                 K_{2,3}-Test((v → w), v, b.alert(w));              // Check for K_{2,3}-subdivision
              if (source of seq ≠ t(ear(v → w))) then Report(K_4); stop;
              else top.end := w; top.SP := seq; top.tail := nil;    // push seq onto stk_{t(ear(v→w))}
                    push top onto stack stk_{t(ear(v→w))};
                    s̃_w := t(ear(v → w));                           // update s̃_w
        ear(v → w) := ear(f);  seq := seq_u;                        // update ear(v → w) and seq
  else
      if (source of seq_u ≠ t(ear(f))) then Report(K_4); stop;
      if (t(ear(f)) = t(ear(v → w)))) then                        // Case (c): seq and seq_u have common source and sink
•         if (∼ (K_{2,3}-found)) then                             // Check for K_{2,3}-subdivision
•              if ((f is not a back-edge) ∧ (s(ear(v → w)) ≠ w)) then K_{2,3}-Test(f, v, b.alert(w));
          if (source of seq ≠ t(ear(v → w))) then Report(K_4); stop;
                                             else seq := PC(seq, seq_u);
          if (ear(f) ⋖ ear(v → w)) then ear(v → w) := ear(f);
      else                                                          // Case (a): t(ear(f)) ≻ t(ear(v → w))
•         if (∼ (K_{2,3}-found)) then
•              if (f is not a back-edge) then K_{2,3}-Test(f, v, b.alert(w));   // Check for K_{2,3}-subdivision
          if (stk_{t(ear(f))} ≠ ∅∧top.end = w)                    // seq_u and top.SP have common terminating vertices
          then top.SP := PC(top.SP, seq_u)                        // merge seq_u with top.SP
          else top.end := w; top.SP := seq_u; top.tail := nil;     // push seq_u onto stack stk_{t(ear(f))}
                push top onto stack stk_{t(ear(f))};
                s̃_w := min_⪯{s̃_w, t(ear(f))};                     // update s̃_w
end; // of Update-ear-of-parent

• Procedure K_{2,3}-Test(e, v, b.alert)
• begin
```

• **if** $(t(ear(e)) \neq v)$ **then** `Report`$(K_{2,3})$; // Theorem 3.5(a) is violated; return a $K_{2,3}$-subdivision
• **else if** $(b.alert)$ **then** `Report`$(K_{2,3})$; // Theorem 3.5(b) is violated; return a $K_{2,3}$-subdivision
• **else** $b.alert := true$; // warning: a non-trivial ear P with $s(P) = v$ has been found; can't have another
• $b := ear(e)$; // for generating a $K_{2,3}$-subdivision when Theorem 3.5(b) is violated
• **end.** // of $K_{2,3}$-`Test`

Lemma 4.1. *Let u be a child of w. Let $f_i, 1 \leq i \leq q$, be the set of incoming back-edges of w such that $t(P_{f_i}) = u_i$ lies on $P_{ear(w\to u)}$ and $u_i \preceq u_{i+1}, 1 \leq i < q$. When the dfs backtracks from u to w, let $SP_{u_i,w}, 1 \leq i \leq q$, be the SP subgraph constructed based on ear P_{f_i}. If there is no ear P_f such that $t(P_f)$ is an internal vertex of $w \rightsquigarrow_T u_q$ and $s(P_f) \prec w$, then on stack stk_w, $SP_{u_i,w}, 1 \leq i < q$, lies above $SP_{u_{i+1},w}$.*

Proof. Since there is no ear P_f such that $t(P_f)$ is an internal vertex of $w \rightsquigarrow_T u_q$ and $s(P_f) \prec w$, therefore, for each $f_i, 1 \leq i < q$, there is no ear P with $t(P) = u_i$ and $s(P) \prec w$. This implies that $SP_{u_i,w}$ is pushed onto stack stk_w after the dfs backtracks to u_i from its child on $P_{ear(w\to u)}$. Hence, $SP_{u_i,w}, 1 \leq i < q$, lies above $SP_{u_{i+1},w}$ on stk_w. □

Theorem 4.2. *In the course of executing **Procedure** `GenCS`, when the dfs backtracks from vertex w to its parent vertex $v(\neq r)$, the parent edge $(w,v)^p$ and the section of ear $P_{ear(v\to w)}$, $s(P_{ear(v\to w)}) \rightsquigarrow_{P_{ear(v\to w)}} w$, including all the ears s*-attached to that section have been transformed into a chain of SP subgraphs, $SP_{w_i \rightsquigarrow w_{i+1}}, 0 \leq i < k$, and $SP_{w_k \rightsquigarrow v}$, where $w_0 = s(P_{ear(v\to w)})$ such that (Figure 7):*

(i) *for each ear P_f with $t(P_f) \in_s P_{ear(v\to w)}$ such that $w_0 \prec s(P_f) \prec w \preceq t(P_f)$, P_f and all the other ears with the same source and sink, and all the ears s*-attached to them have been transformed into an SP subgraph $SP_{w_i,s(P_f)}$, for some $i, 1 \leq i \leq k$, such that on stack $stk_{s(P_f)}$, there is an entry **x** with **x**$.end = w_i$, **x**$.SP = SP_{w_i,s(P_f)}$ and*

$$\boldsymbol{x}.tail = \begin{cases} SP_{w_{i-1} \rightsquigarrow w_i}, & \textit{if } t(f) = \tilde{s}_i, \textit{ where } \tilde{s}_i = \min_{\preceq}\{t(f') \mid (f' \in E \setminus E_T) \wedge t(P_{f'}) = w_i\}; \\ nil, & \textit{otherwise.} \end{cases}$$

(ii) $\forall w_i, 1 \leq i \leq k, \exists SP_{w_i,s(P_f)}$*, with* $w_0 \prec s(P_f) \prec w$*;*

(iii) $seq = SP_{w_k \rightsquigarrow v}$.

Proof. (By induction on the height of w in T) When w is a leaf, based on the discussion given before **Algorithm** `SP&Outerplanar` above, it is easily verify that the theorem holds for w.

Let w be an internal vertex of T and $(w,v)^p \in E_s$. We shall call the chain of SP subgraphs satisfying Conditions $(i) - (iii)$ the w-SPchain. Let $|\overline{\underline{\ell \| u}}|$ be the next node in $L[w]$ such that $u \neq v$.

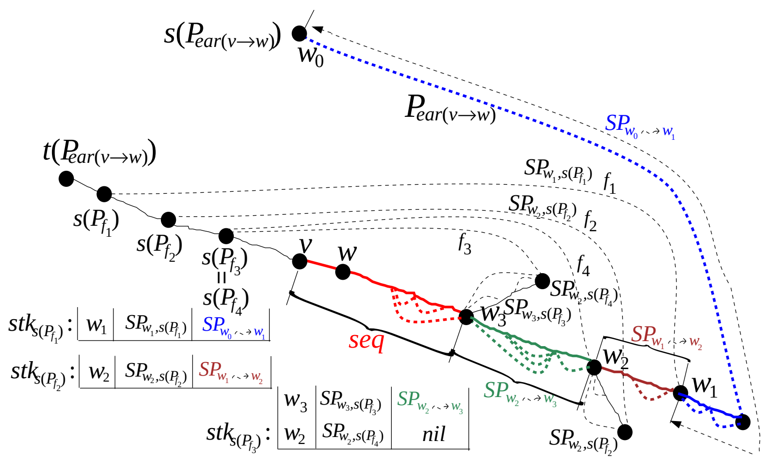

Figure 7: A chain of SP subgraphs associated with vertex w

(i) If u is unvisited, then u becomes a child of w. When the dfs backtracks from u to w, by the induction hypothesis, the u-SPchain has been created. Let it be $SP_{u_i \rightsquigarrow u_{i+1}}, 0 \leq i < h$, and $(seq_u =) SP_{u_h \rightsquigarrow w}$, where $u_0 = s(P_{ear(w \to u)})$. Procedure `Update-seq` is then invoked to pop stack stk_w. Let $q, 1 \leq q \leq h$ be the smallest index such that $SP_{u_q,w}$ exists. If there is an $SP_{u_j,z}$, $q < j \leq h$, such that $z \prec w$, let it be the one closest to w (Figure 6(a)). Let u_m be closest to u_j such that $m < j$ and $SP_{u_m,w}$ exists. Since $z \prec w$, $SP_{u_j,z}$ is not on stk_w. Therefore, $s(\mathbf{u_i}.tail) \preceq u_j$ for every entry $\mathbf{u_i}$ above entry $\mathbf{u}_m$ on stk_w. Hence, after all the entries above $\mathbf{u}_m$ are popped, seq_u becomes an $SP_{u_j \rightsquigarrow w}$. When $\mathbf{u}_m$ is popped, as $\mathbf{u}_m.end = u_m \neq u_j$, a K_4-subdivision is returned. On the other hand, if there is no $SP_{u_j,z}$, $q < j \leq h$, such that $z \prec w$, then by Condition (ii), $SP_{u_i,w}, q \leq i \leq h$, exist. By Lemma 4.1, on stack stk_w, $SP_{u_i,w}, q < i \leq h$, lies above $SP_{u_{i+1},w}$. Hence, when stk_w is emptied, seq_u is updated to an $SP_{u_{\hat{q}} \rightsquigarrow w}$, where $\hat{q} \in \{q, q-1\}$ (depending on whether $\exists SP_{u_q,z}$ with $z \prec w$), and the u-SPchain becomes $SP_{u_i \rightsquigarrow u_{i+1}}, 0 \leq i < \hat{q}$, and $(seq_u =) SP_{u_{\hat{q}} \rightsquigarrow w}$. It is easily verified that the modified u-SPchain satisfies Conditions $(i) - (iii)$.

(a) If $t(ear(v \to w)) \prec t(ear(w \to u))$, then seq_u terminates at w. If $\hat{q} \neq 0$, then there is an ear $\tilde{P}$ with $t(\tilde{P}) = u_{\hat{q}}$ such that $u_0 \prec s(\tilde{P}) \prec w$ (Figure 6(b)). This ear violates Condition (a) of Theorem 3.4. The algorithm thus terminates execution and returns a K_4-subdivision. Otherwise, the u-SPchain consists of just $seq_u (= SP_{u_0 \rightsquigarrow w} = SP_{w,u_0})$. If on stack stk_{u_0}, $\mathbf{top}.end = w$, then $\mathbf{top}.SP$ is replaced by $\mathtt{PC}(\mathbf{top}.SP, seq_u)$ as the two SP subgraphs have common source and sink. Otherwise, seq_u is pushed onto stack stk_{u_0} such that $\mathbf{top}.SP = seq_u$, $\mathbf{top}.end = w$ and $\mathbf{top}.tail = nil$. (b) If $t(ear(w \to u)) \prec t(ear(v \to w))$, let the current w-SPchain be $SP_{w_i \rightsquigarrow w_{i+1}}, 0 \leq i < p$, and $(seq =) SP_{w_p \rightsquigarrow w}$. Then seq terminates at w. As with Case (a), $p = 0$ or a violation of Condition (a) of Theorem 3.4 is detected. In the former case, the current w-SPchain consists of just $seq(= SP_{w_0 \rightsquigarrow w} = SP_{w,w_0})$ which is pushed onto stack stk_{w_0} such that **top**.$SP = seq$, **top**.$end = w$ and **top**.$tail = nil$. The u-SPchain then becomes the current w-SPchain and $ear(v \to w) := ear(w \to u), seq := seq_u$. (c) If $t(ear(w \to u)) = t(ear(v \to w))$ (i.e. $u_0 = w_0$), then as with Cases (a) and (b), $\hat{q} \neq 0$ or $p \neq 0$ implies that there is an ear P with $t(P) = u_{\hat{q}}$ or $t(P) = w_p$ whereby violating Condition (a) of Theorem 3.4. The algorithm thus terminates execution and return a K_4-subdivision. Otherwise, $seq = SP_{w_0 \rightsquigarrow w}$ and $seq_u = SP_{u_0 \rightsquigarrow w}$ which are merged by $seq := \mathsf{PC}(seq, seq_u)$, and the current w-SPchain consists of just $SP_{w_0 \rightsquigarrow w}(= seq)$. Furthermore, if $ear(w \to u) \lessdot ear(v \to w)$, then $ear(v \to w) := ear(w \to u)$.

(ii) If u is visited, the u-SPchain is $SP_{u_0 \rightsquigarrow w}$ consisting of $(u \curvearrowleft w)^\ell$ and $seq_u = SP_{u_0 \rightsquigarrow w}$. The remaining argument is same the above case where u is unvisited.

When $L[w]$ is completely processed, let the current w-SPchain be $SP_{w_i \rightsquigarrow w_{i+1}}, 0 \leq i < k$, and $(seq =)SP_{w_k \rightsquigarrow w}$. The chain consists of $s(P_{ear(v \to w)}) \rightsquigarrow_{P_{ear(v \to w)}} w$ and all the ears s^*-attached to it.

For each ear P_f with $t(P_f) \in_s P_{ear(v \to w)}$ such that $w_0 \prec s(P_f) \prec w \preceq t(P_f)$, if $t(P_f) \neq w$, then P_f satisfies Conditions (i) by the induction hypothesis. If $t(P_f) = w$, then from the above discussion, on stack $stk_{s(P_f)}$, **top**.$SP = SP_{w,s(P_f)}$ and **top**.$tail = nil$. Furthermore, by a simple induction on i, where i is the number of nodes in $L[w]$ that have been processed, it is easily verified that $\tilde{s}_w = \min_{\preceq}\{t(f') \mid (f' \in E \setminus E_T) \wedge (t(P_{f'}) = w)\}$. Hence, after **top**.$tail := seq(= SP_{w_k \rightsquigarrow w})$ on $stk_{\tilde{s}_w}$, P_f satisfies Conditions (i). Condition (ii) is satisfied by the induction hypothesis and the existence of $SP_{w,s(P)}$ if there exists P with $t(P) = w$. Finally, as $seq = nil$ if $\exists P$ with $t(P) = w$, and $seq = (w_k \rightsquigarrow w)$, otherwise, after $seq := \mathsf{SC}(seq, (w, v)^p)$, Condition (iii) clearly holds. Hence, when the *dfs* backtracks from w to v, the w-SPchain has been correctly generated. ☐

Theorem 4.3. *Algorithm SP&Outerplanar generates a construction sequence for G if G is SP or a K_4-subdivision of G, otherwise, in $O(|V|)$ time.*

Proof. If G is not SP, then as was explained in the proof of Theorem 4.2, a violation of Condition (a) or (b) of Theorem 3.4 will be detected and a K_4-subdivision is returned. Otherwise, let w be the child of the root r (G is biconnected implies that w is unique). If w has no children, then G consists of a set of p parallel edges with end-vertices r and w. When the *dfs* reaches w, $seq = nil$. Since $v = r$, therefore, $seq = \mathsf{PC}(seq, (w, v)^p) = (w, v)^p$ which is a construction sequence of G when the *dfs* backtracks to r. If w has children. Let u be the child of w lying on the ear P_1 (i.e. $P_{ear(w \to u)} = P_1$). Since $\nexists P_f$ with

$r \prec s(P_f) \prec w$, when the depth-first search backtracks from u to w, Theorem 4.2(ii) implies that after stk_w is emptied the u-SPchain consists of solely $seq_u = SP_{r \rightsquigarrow w}$. It follows that the current w-SPchain consists of $seq = SP_{r \rightsquigarrow P_{ear(w \to u)}} w$. Similarly, for each remaining child u' of w, the u'-SPchain consists of just $seq_{u'} = SP_{r \rightsquigarrow P_{ear(w \to u')}} w$. Since seq and $seq_{u'}$ have common source r and sink w, $seq_{u'}$ is merged into seq by $seq = \texttt{PC}(seq, seq_{u'})$. Therefore, after $L[w]$ is completely processed, the current w-SPchain consists of $seq = SP_{r \rightsquigarrow w}$ and the final $\texttt{PC}((r,w)^p, seq)$ produces a construction sequence for G.

The initialization clearly takes $O(|V|)$ time. $\forall e \in E \backslash E_T$, since $ear(e) = e$, determining $ear(e)$ takes $O(1)$ time. Determining $ear(e), \forall e \in E_T$, takes $O(|V|)$ time during the dfs. By storing $ear(e), e \in E_T$, as $ear[w]$, where $e = (parent(w) \to w)$, in an array $ear[1..|V|]$, retrieving $ear(e)$, $s(ear(e))$ and $t(ear(e))$ takes $O(1)$ time each. By representing every SP subgraph with a decomposition tree that keeps its source and sink at the root node, retrieving the source and sink of an SP subgraph, and performing $\texttt{SC}(S_1, S_2)$, $\texttt{PC}(S_1, S_2)$ and determining their respective source and sink each takes $O(1)$ time. Hence, excluding the time spent on generating a K_4-subdivision, Procedure `Update-ear-of-parent` takes $O(1)$ time and the **while** loop in Procedure `Update-seq` takes $O(1)$ time per iteration.

For each $w \in V$, The initialization steps in Procedure `GenCS` take $O(1)$ time. The body of the **for** loop excluding the call to Procedure `Update-seq` and the recursive call (which is charged to vertex u) takes $O(1)$ time. Procedure `Update-seq` processes the entries on stack stk_w. Since each entry on the stack corresponds to a distinct incoming back-edge of w and the **while** loop takes $O(1)$ time per iteration, the total time spent on Procedure `Update-seq` for vertex w is thus $O(deg_G(w))$. The **for** loop thus takes $\sum_{u \in L[w]} O(1) + O(deg_G(w)) = O(deg_G(w))$ time. The **if** statements following the **for** loop takes $O(1)$ time. Hence, **Algorithm** SP&Outerplanar takes $\sum_{w \in V} O(deg_G(w)) = O(|V|)$ time to generate an SP construction sequence if G is an SP graph.

If G is not an SP graph, as will be shown in Section 4.3.1, generating a K_4-subdivision involves tracing out at most three ears and a tree path which takes $O(|V|)$ time. Hence, the algorithm takes $O(|V|)$ time. $\square$

4.2 Recognizing Outerplanar graphs

The following lemma shows that the dfs-tree T of an outerplanar graph has a very simple structure.

Lemma 4.4. *If G is outerplanar, every vertex has at most two children in T.*

Proof. An immediate consequence of Theorem 3.5 (a) and (b). $\square$

Since G is outerplanar if and only if its underlying simple graph is outerplanar, in the algorithm presented below, we disregard the parallel-edge count ℓ in the nodes of the adjacency lists.

The algorithm is conceptually very simple. The exterior boundary is constructed during the depth-first search in a bottom-up manner by starting from the leaves and gradually moving towards the root.

In general, at each leaf w, the parent edge of w and the lexicographically smallest outgoing back-edge of w are added to the exterior boundary.

At each internal vertex w, owing to Lemma 4.4, only two cases are to be considered.

1. w has one child: let the child be u and $(z \curvearrowleft w)$ be the lexicographically smallest outgoing back-edge of w. If $(z \curvearrowleft w)$ exists and $(z \curvearrowleft w) \lessdot ear(w \to u)$, then $(z \curvearrowleft w)$ is added to the exterior boundary; otherwise, the parent edge of w is added.

2. w has two children: no edge incident on w is added to the boundary at w. Note that, however, two of such edges must have been added to the boundary at some descendants of w.

Clearly, the above method for determining the exterior boundary can be carried out concurrently with the construction of the SP construction sequence. In **Algorithm** SP&Outerplanar, the instructions for constructing the exterior boundary are marked with a •. The flags $K_{2,3}$-$found$ and $b.alert(w)$ are used to detect violation of Conditions (a) and (b) of Theorem 3.5. In Procedure `Update-ear-of-parent`, the newly inserted instructions are for detecting violation of the two conditions. Specifically, when a non-trivial ear terminating at w is found, if the other terminating vertex of the ear is not $parent(w)$, a violation of Condition (a) is detected. If $parent(w)$ is the other terminating vertex but $b.alert(w)$ is $true$, a violation of Condition (b) is detected; otherwise, $b.alert(w)$ is set to $true$. Note that detecting violation of Condition (c) is taken care of by the part of the algorithm that constructs the decomposition tree.

Lemma 4.5. *In the course of executing **Procedure** `GenCS`, $\forall w \in V \backslash \{r\}$, when the depth-first search backtracks from vertex w to its parent vertex v, let G_w be the subgraph of G consisting of:*

- *the cycle formed by the ear $P_{ear(v \to w)}$ and the tree-path $s(P_{ear(v \to w)}) \rightsquigarrow_T t(P_{ear(v \to w)})$, and*
- *$\mathcal{P}_w = \{P \mid (P$ is a non-trivial ear $) \wedge (w \preceq t(P))\}$.*

Let E_B^w be the set of edges added to E_B while the dfs was traversing T_w. Then E_B^w and $s(P_{ear(v \to w)}) \rightsquigarrow_T v$ form the exterior boundary of an outerplanar embedding of G_w.

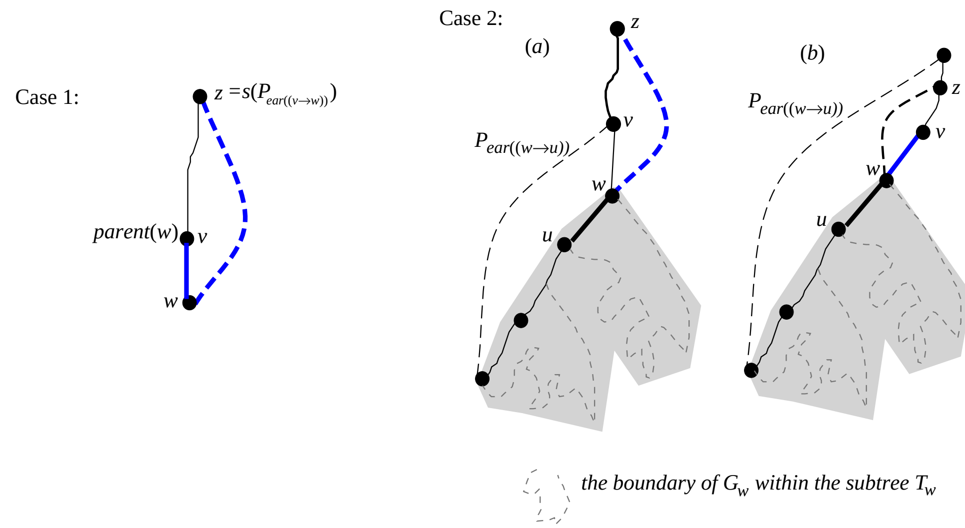

Figure 8: Case 1: w has no child. Case 2: w has exactly one child.

Proof. (By induction on the height of w in T) When w is a leaf, as $\mathcal{P}_w = \emptyset$, G_w is a cycle consisting of $P_{ear(v \to w)}$ and $s(P_{ear(v \to w)}) \rightsquigarrow_T t(P_{ear(v \to w)})$. Since $(v \to w)$ and $ear(v \to w)$ are the two edges added to E_B, where $ear(v \to w) = (z \curvearrowleft w)$ is the lexicographically smallest outgoing back-edge of w, $E_B^w = \{(v \to w), (z \curvearrowleft w)\}$ (Figure 7, Case 1). As $(v \to w), (z \curvearrowleft w)$ and $z \rightsquigarrow_T v$ form G_w, and $(s(P_{ear(v \to w)}) \rightsquigarrow_T v) = (z \rightsquigarrow_T v)$, the lemma holds for w.

When w is an internal vertex of T, first, consider the case where w has exactly one child u. Let $(z \curvearrowleft w)$ be the lexicographically smallest outgoing back-edge of w. If $z \prec t(ear(w \to u))$, then $t(P_{ear(w \to u)}) = w$ (Figure 7, Case 2(a)). If $s(P_{ear(w \to u)}) \neq v$, a violation of Condition (a) of Theorem 3.4 is detected, and the algorithm terminates execution and returns a $K_{2,3}$-subdivision. Otherwise, G_u consists of $P_{ear(w \to u)}$, $(v \to w)$ and $\mathcal{P}_u$. Since G_w consists of $P_{ear(v \to w)}, s(P_{ear(v \to w)}) \rightsquigarrow_T t(P_{ear(v \to w)})$ and $\mathcal{P}_w$; $P_{ear(v \to w)}$ and $s(P_{ear(v \to w)}) \rightsquigarrow_T t(P_{ear(v \to w)})$ are equivalent to $(z \curvearrowleft w), z \rightsquigarrow_T v$, and $v \to w$, it follows that G_w consists of $(z \curvearrowleft w), z \rightsquigarrow_T v, v \to w$, and $\mathcal{P}_w$. Then $\mathcal{P}_w = \mathcal{P}_u \cup \{P_{ear(w \to u)}\}$ implies that G_w consists of $(z \curvearrowleft w), z \rightsquigarrow_T v, v \to w, \mathcal{P}_u$ and $P_{ear(w \to u)}$ which implies that G_w consists of $(z \curvearrowleft w), z \rightsquigarrow_T v$ and G_u. By the induction hypothesis, E_B^u and $(v \to w)$ form the exterior boundary of an outerplanar embedding of G_u. Therefore, by embedding $(z \curvearrowleft w)$ and $z \rightsquigarrow_T v$ onto the exterior face of the planar embedding of G_u and connecting them to the latter at vertices w and v, we obtain an outerplanar embedding of G_w. Since $ear(v \to w)(= (z \curvearrowleft w))$ was added to E_B^u at w, $E_B^w = E_B^u \cup \{(z \curvearrowleft w)\}$. This implies that E_B^w and $z \rightsquigarrow_T v (= s(P_{ear(v \to w)}) \rightsquigarrow_T v)$ form the exterior boundary of an outerplanar embedding of G_w.

On the other hand, if $(z \curvearrowleft w)$ does not exist or $t(ear(w \to u)) \preceq z$, then $ear(v \to w) = ear(w \to u)$

Case 3:

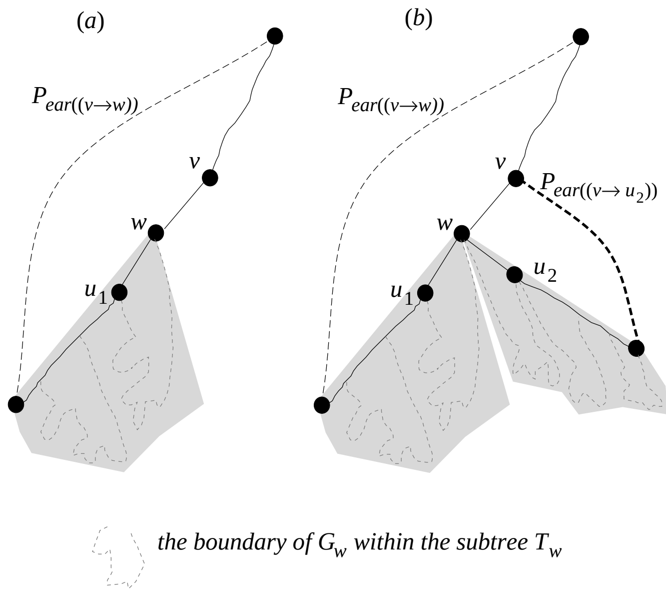

Figure 9: w has exactly two children.

which implies that $P_{ear(v\to w)} = P_{ear(w\to u)}$ (Figure 7, Case $2(b)$). Therefore, G_w consists of $P_{ear(v\to w)}$, $s(P_{ear(v\to w)}) \rightsquigarrow_T t(P_{ear(v\to w)})$ and $\mathcal{P}_w$ implies that G_w consists of $P_{ear(w\to u)}$, $s(P_{ear(w\to u)}) \rightsquigarrow_T t(P_{ear(w\to u)})$ and $\mathcal{P}_w$. Since w has only one child, therefore $\mathcal{P}_w = \mathcal{P}_u$ which implies that G_w consists of $P_{ear(w\to u)}$, $s(P_{ear(w\to u)}) \rightsquigarrow_T t(P_{ear(w\to u)})$ and $\mathcal{P}_u$. It follows that $G_w = G_u$. Since by the induction hypothesis, G_u has an outerplanar embedding, G_w thus has an outerplanar embedding. Moreover, by the induction hypothesis, E_B^u and $s(P_{ear(w\to u)}) \rightsquigarrow_T w$ form the exterior boundary of the outerplanar embedding of G_u and hence of G_w. As $E_B^w = E_B^u \cup \{(v \to w)\}$ and $s(P_{ear(w\to u)}) \rightsquigarrow_T w$ consists of $s(P_{ear(v\to w)}) \rightsquigarrow_T v$ and $(v \to w)$, E_B^w and $s(P_{ear(v\to w)}) \rightsquigarrow_T v$ form the exterior boundary of the outerplanar embedding of G_w. The assertion thus holds for w.

Next, consider the case where w has exactly two children u_1 and u_2 such that $ear(w \to u_1) \lessdot ear(w \to u_2)$. Then G_{u_1} consists of $P_{ear(w\to u_1)}$, $s(P_{ear(w\to u_1)}) \rightsquigarrow_T t(P_{ear(w\to u_1)})$ and $\mathcal{P}_{u_1}$. Similarly, G_{u_2} consists of $P_{ear(w\to u_2)}$, $s(P_{ear(w\to u_2)}) \rightsquigarrow_T t(P_{ear(w\to u_2)})$ and $\mathcal{P}_{u_2}$ (Figure 8). Since $ear(w \to u_1) \lessdot ear(w \to u_2)$, $t(P_{ear(w\to u_2)}) = w$ and $s(P_{ear(w\to u_2)}) = v$, or a violation of Condition (a) of Theorem 3.5 is detected. Therefore, G_{u_2} consists of $P_{ear(w\to u_2)}$, $v \to w$ and $\mathcal{P}_{u_2}$. Moreover, $P_{ear(v\to w)} = P_{ear(w\to u_1)}$.

Now, G_w consists of $P_{ear(v\to w)}$, $s(P_{ear(v\to w)}) \rightsquigarrow_T t(P_{ear(v\to w)})$ and $\mathcal{P}_w$. Since $P_{ear(v\to w)} = P_{ear(w\to u_1)}$ and $\mathcal{P}_w = \mathcal{P}_{u_1} \cup P_{u_2} \cup \{P_{ear(w\to u_2)}\}$, it follows that G_w consists of $P_{ear(w\to u_1)}, s(P_{ear(w\to u_1)}) \rightsquigarrow_T t(P_{ear(w\to u_1)}), \mathcal{P}_{u_1}, \mathcal{P}_{u_2}$ and $P_{ear(w\to u_2)}$ or equivalently, G_{u_1}, and G_{u_2} excluding $v \to w$.

By the induction hypothesis, $E_B^{u_1}$ and $s(P_{ear(w\to u_1)}) \rightsquigarrow_T w$ form the exterior boundary of an outerplanar embedding of G_{u_1}; $E_B^{u_2}$ and $v \to w$ form the exterior boundary of an outerplanar embedding of G_{u_2}. By embedding G_{u_2} onto the exterior face of the planar embedding of G_{u_1} and connecting the two plane graphs at vertices v and w, we obtain a planar embedding of G_w. Since $E_B^w = E_B^{u_1} \cup E_B^{u_2}$, this immediately implies that E_B^w and $s(P_{ear(v\to w)}) \rightsquigarrow_T v$ form the exterior boundary of an outerplanar embedding of G_w. The lemma thus holds for w. □

Theorem 4.6. *Algorithm SP&Outerplanar generates the exterior boundary of an outerplanar embedding of G if G is outerplanar, or a K_4-subdivision or $K_{2,3}$-subdivision of G, otherwise, in $O(|V|)$ time.*

Proof. If G is not outerplanar, then as was explained in the proof of Theorem 4.5, a violation of Condition (a) or (b) of Theorem 3.5 will be detected (violation of Condition (c) is taken care of by that part of the algorithm for SP graphs). Otherwise, let w be the child of the root r. Then $P_{ear(r\to w)} = P_1$. By Lemma 4.5, when the dfs backtracks from w to r, $E_B^w(= E_B)$ forms the exterior boundary of an outerplanar embedding of G_w. Since G_w consists of P_1 and $\mathcal{P}_w$, G and G_w differ in only the trivial ears which can be embedded into the interior faces of the outerplanar embedding of G_w because no violation of Condition (c) was detected. Hence E_B is the edge set of the exterior boundary of an outerplanar embedding of G.

The initialization related to outerplanar testing clearly takes $O(|V|)$ time. In **Procedure** `GenCS`, since the instructions for testing outerplanarity (marked by •'s) take $O(1)$ time for each $w \in V \setminus \{r\}$, **Procedure** `GenCS`, excluding the time spent on detecting $K_{2,3}$-subdivision, takes $O(|V|)$ time. To detect $K_{2,3}$-subdivision, the instructions in **Procedure** `Update-end-of-parent` (marked by •'s) take $O(1)$ time for each $u \in L[w]$. **Procedure** $K_{2,3}$`-Test` excluding the time spent on Report($K_{2,3}$) takes $O(1)$ for each $e \in E_T$. Detecting $K_{2,3}$-subdivision thus takes a total of $O(|V|)$ time. As will be shown in Section 4.3.2, generating a $K_{2,3}$-subdivision involves tracing out at most two ears and two tree-paths. **Procedure** `Report` thus takes $O(|V|)$ time. Detecting K_4-subdivision is taken care of when the algorithm is checking if G is series-parallel. Hence, **Algorithm** `SP&Outerplanar` takes $O(|V|)$ time on outerplanarity testing. □

4.3 Generating forbidden subgraphs

4.3.1 Generating a K_4-subdivision

When a violation of Condition (a) of Theorem 3.4 is detected in **Procedure** `Update-ear-of-parent`, if it is caused by the condition 'source of $seq_u(= u_h) \neq t(ear(f))$', then $f = (w \to u)$ and **Procedure**

`Report` is called to generate a K_4-subdivision consisting of (Figure 6(b)):

- $P_{ear(w\to u)}$ and the lexicographically smallest ear $\tilde{P}$ with $t(\tilde{P}) = u_h \wedge s(P_{ear(w\to u)}) \prec s(\tilde{P})$;
- $s(P_{ear(v\to w)}) \rightsquigarrow_{P_{ear(v\to w)}} w$ and $s(P_{ear(v\to w)}) \rightsquigarrow_T w$.

The ear $P_{ear(w\to u)}$ can be generated by starting from the back-edge $ear(w \to u)$, using the array $parent[z], \forall z \in V$, to determine the tree-edges on it until vertex w is reached. The path $s(P_{ear(v\to w)}) \rightsquigarrow_{P_{ear(v\to w)}} w$ can be generated similarly. The path $s(P_{ear(v\to w)}) \rightsquigarrow_T w$ can be generated similarly by starting from w. To determine $\tilde{P}$, we determine $\tilde{e}$ such that $ear(\tilde{e}) = \min_{\lessdot}\{ear(e) \mid (e = (u_h \to y) \vee e = (y \curvearrowleft u_h)) \wedge (t(ear(w \to u)) \prec t(ear(e)))\}$. Then $\tilde{P} = P_{ear(\tilde{e})}$ which can be generated similar to $P_{ear(w\to u)}$. Since all of the above steps take $O(|V|)$ time, The K_4-subdivision can be constructed in $O(|V|)$ time.

If it is caused by the condition 'source of $seq(= w_k) \neq t(ear(v \to w))$', then **Procedure** `Report` is called to generate a K_4-subdivision consisting of:

- $P_{ear(v\to w)}$ and the lexicographically smallest ear $\tilde{P}$ with $t(\tilde{P}) = w_k \wedge s(P_{ear(v\to w)}) \prec s(\tilde{P})$;
- $s(P_{ear(f)}) \rightsquigarrow_{P_{ear(f)}} w$ and $s(P_{ear(f)}) \rightsquigarrow_T w$.

As with the above case, the K_4-subdivision can be constructed in $O(|V|)$ time.

When a violation of Condition (b) of Theorem 3.4 is detected in **Procedure** `Update-seq`, **Procedure** `Report` is called to generate a K_4-subdivision consisting of (Figure 6(a)):

- P and $s(P) \rightsquigarrow_T t(P)$, such that $P = P_{ear(e_h)}$, where e_h is the parent edge of **top**.end.
- an ear $\tilde{P}$ with $t(\tilde{P}) = s(seq) \wedge s(\tilde{P}) \prec w$,
- an ear P_f with $s(P_f) = w$ and $t(P_f) =$ **top**.SP, where $f = ear(e)$ for some e incident to **top**.end.

It is easily verified that the K_4-subdivision can be constructed in $O(|V|)$ time.

4.3.2 Generating a $K_{2,3}$-subdivision

When a violation of Condition (a) of Theorem 3.5 is detected in **Procedure** `Update-ear-of-parent`, **Procedure** `Report` is called to generate a $K_{2,3}$-subdivision consisting of (Figure 4(a)):

- $P_{ear(v\to w)}$ and $P_{ear(w\to u)}$,
- $\begin{cases} s(P_{ear(v\to w)}) \rightsquigarrow_T t(P_{ear(v\to w)}), & \text{if } ear(v \to w) \lessdot ear(w \to u); \\ s(P_{ear(w\to u)}) \rightsquigarrow_T t(P_{ear(w\to u)}), & \text{otherwise} \quad \text{(in Figure 4(a), } P_{ear(w\to u)} = P_j\text{; disregard } u\text{).} \end{cases}$

It is easily verified that the aforementioned ears and tree-paths can be generated in $O(|V|)$ time.

When a violation of Condition (b) of Theorem 3.5 is detected in **Procedure** `Update-ear-of-parent`,

Procedure `Report` is called to generate a $K_{2,3}$-subdivision consisting of (Figure 4(b)):

- $\begin{cases} P_{ear(w\to u)}, P_b, s(P_{ear(v\to w)}) \rightsquigarrow_T v, s(P_{ear(v\to w)}) \rightsquigarrow_{P_{ear(v\to w)}} w, & \text{if } ear(v\to w) \lessdot ear(w\to u); \\ P_{ear(v\to w)}, P_b, s(P_{ear(w\to u)}) \rightsquigarrow_T v, s(P_{ear(w\to u)}) \rightsquigarrow_{P_{ear(w\to u)}} w, & \text{otherwise.} \end{cases}$

Similar to the above case, the $K_{2,3}$-subdivision can be constructed in $O(|V|)$ time.

5 Authentication of the certificates

5.1 Positive certificate

Construction sequence: To authenticate the construction sequence, we use it to construct adjacency lists $\hat{L}(v), v \in V$, of G. If $\hat{L}$ is identical to the original (non-compact) adjacency lists $\tilde{L}$ of G, we confirm G is series-parallel. Otherwise, the certificate is rejected. This is done by traversing the decomposition tree in post-order as follows:

At each vertex v, let $blk_v = false$ if v can be the source or sink of an SP subgraphs, v_- be the number of SP subgraphs constructed thus far with v as the source, v_+ be the number of SP subgraphs constructed as thus with v as the sink. Initially, $blk_v = false, v_- = v_+ = 0$. During the traversal, on encountering:

- a leaf node $|\overline{\underline{\ell\|u|\mathbf{e}|v}}|$: if blk_u or blk_v is *true*, reject the certificate (edge e cannot be merged with existing SP subgraphs). Otherwise, $u_- := u_- + 1, v_+ := v_+ + 1$ indicating the number of SP-subgraphs with u as source (t as sink, respectively) is increased by one; add ℓ u-nodes to $\hat{L}[v]$, ℓ v-nodes to $\hat{L}[u]$;
- an internal node $|\overline{\underline{0\|s|\mathbf{S}|t}}|$: let $|\overline{\underline{i\|s|\chi|w}}|$ and $|\overline{\underline{j\|w|\chi|t}}|$ ($\chi \in \{\mathbf{S},\mathbf{P},\mathbf{e}\}$) be its left and right child, respectively. Let $blk_w := true$ (w can no longer be a source or sink after this `SC` operation). If $w_- \neq 1$ or $w_+ \neq 1$, then reject the certificate (there remains SP subgraphs having w as source or sink which cannot be merged into the graph under construction); otherwise, $w_- := w_+ := 0$;
- an internal node $|\overline{\underline{0\|s|\mathbf{P}|t}}|$: let $s_- := s_- - 1, t_+ := t_+ - 1$ indicating the number of SP-subgraphs with s as source (t as sink, respectively) is decreased by one.

When the traversal terminates at the root node, let the root node be $|\overline{\underline{0\|r|\mathbf{P}|s}}|$. If $not(r_- = s_+ = 1)$, or $not(r_+ = s_- = 0)$, or $not(v_- = v_+ = 0), \forall V \setminus \{r,s\}$, reject the certificate as it generated a disconnected graph which cannot be G. Otherwise, use radix sort to sort both $\hat{L}(v)$, and $\tilde{L}[v], v \in V$, and then compare them. Confirm G is series-parallel if they are identical, reject the certificate otherwise. This authentication procedure clearly takes $O(|E|)$ time. Its correctness is easily verified.

Although an $SP_{x,y}$ has x as source and y a sink, as it is constructed based on an ear with source y and sink x, its source and sink are y and x, respectively, upon completion. Fortunately, we do not need to swap the source and sink physically for each node in its decomposition tree. What we need is to mark the root node indicating that it is a root node and maintain a switch $swap$. Initially, $swap := false$. During the traversal, when a marked node is entered, let $swap := not(swap)$. Then every node $|\ell\|u|\chi|v|$ in the corresponding decomposition tree is treated as v is the source and u is the sink if and only if $swap = true$. When the traversal backtracks from a marked node, let $swap := not(swap)$. Let $SP_{x_1,y_1}, SP_{x_2,y_2}, \ldots, SP_{x_i,y_i}, \ldots SP_{x_k,y_k}$ such that $SP_{x_{i+1},y_{i+1}}$ is s-attached to $SP_{x_i,y_i}, 1 \leq i < k$. Then $swap = fasle$ if and only if i is odd.

Exterior boundary: Let the exterior boundary be $C : w_1, w_2, \ldots, w_m, w_1$. Based on C, a *dfs* is performed over G to make the path $w_1 w_2 \ldots w_m$ a *dfs* tree of G which includes verifying $|V| = m$ and every vertex in G appears exactly once in C. This takes $O(|E|)$ time. In building the *dfs* tree, a ear-decomposition, $P_i, 1 \leq i \leq |E| - |V| + 1$, of G is created such that P_1 is the cycle C and each $P_i, i > 1$, is a trivial ear (i.e. a back-edge) s-attached to C. Then C is the exterior boundary of G if and only if the trivial ears $P_i, 2 \leq i \leq |E| - |V| + 1$, can all be embedded into the interior face of C if and only if no two of them are interlacing. The last condition can be verified using the method for detecting K_4-subdivision in SP graphs (see Section 4.1). Since the ears are all trivial and no construction sequence is to be generated, seq and the SP subgraphs stored on the stacks need not be represented by decomposition trees but just by their *source* and *sink*. The correctness is obvious. The authentication of the exterior boundary thus takes $O(|E|)$ time.

5.2 Negative certificate

Since a K_4-subdivision consists of six vertex-disjoint paths sharing four terminating-vertices, we first verify that there are exactly four distinct terminating vertices each of which is a common terminating vertex of three paths and that no two paths have more then one common terminating vertices. This can be done in $O(1)$ time. Then, each path is traced using the adjacency lists and every internal vertex encountered is marked to verify that the edges on the path are edges of G and the paths are vertex-disjoint except at their terminating vertices. This can clearly be done in $O(|E|)$ time. Hence, verifying that the six paths are in G takes $O(|E|)$ time. Verifying a $K_{2,3}$-subdivision can be done similarly in $O(|E|)$ time.

6 Non-biconnected graphs

6.1 Graphs with predesignated source and sink for SP graphs

We observed that if G is SP and $(r, s)^p$ is the first edge **Algorithm** `SP&Outerplanar` uses to traverse G, the decomposition tree generated will make r the source and s the sink and the last composition operation performed is $\texttt{PC}((r, s)^p, seq)$, where seq is a construction sequence of the SP subgraph $G\backslash\{(r, s)\}$. Hence, determining if G is SP with predesignated source u and sink v can be reduced to determining if $G \cup \{(u, v)\}$ is SP with source u and sink v by starting the dfs with the edge (u, v). If the algorithm reports that $G \cup \{(u, v)\}$ is not a SP graph with u as source and v as sink, then so is not G. Otherwise, seq is a construction sequence of G with u and v as the source and sink, respectively.

6.2 SP graphs

If G is not biconnected, since each biconnected component of G is connected to other biconnected components through the cut-vertices it contains and the cut-vertices must be its terminals if G is SP, it is easily verified that the biconnected components of G can be connected as a chain $B_i, 1 \le i \le h$, such that B_i and B_{i+1} share a common cut-vertex c_i, and $c_i, 1 \le i < h$, are distinct. Therefore, we can decompose G into its biconnected components, construct a decomposition tree for each biconnected component, and then join the decomposition trees with the $\texttt{SC}$ operation into a decomposition tree of G. Note that for $B_i, 2 \le i \le h - 1$, the source and sink must be c_i and c_{i-1}, respectively. The method described in Section 6.1, i.e. running the algorithm on $B_i \cup \{(c_i, c_{i-1})\}$, can be used. For B_1, the source must be c_1 but the sink can be any vertex adjacent to c_1; for B_h, the source can be any vertex adjacent to c_{h-1} but the sink must be c_{h-1} .

For $B_i, 2 \le i \le h - 1$, if a K_4-subdivision containing (c_i, c_{i-1}) is returned as a negative certificate and (c_i, c_{i-1}) is not an edge of B_i, then the K_4-subdivision is a negative certificate for $B_i \cup \{(c_i, c_{i-1})\}$ but not for B_i as it is not a subgraph of B_i. Should that be the case, we return the K_4-subdivision after the edge (c_i, c_{i-1}) is removed as a negative certificate for B_i. This is justified by the following characterization theorem for SP graph with designated source and sink.

Theorem 6.1. [3] *A biconnected graph is not SP with source s and sink t if it contains a subdivision of $\Theta_4^{s,t}$, where $\Theta_4^{s,t}$ results from a K_4 after the edge connecting two of its vertices s and t is removed.*

6.3 Outerplanar graphs

Theorem 6.2. [8] *A graph is outerplanar if and only if each of its biconnected components is outerplanar.*

If $B_i \cup \{(c_i, c_{i-1})\}, 2 \leq i < h$, is SP with source c_i and sink c_{i-1}, then $B_i \cup \{(c_i, c_{i-1})\}$ is outerplanar if and only if B_i is outerplanar. The only if part is obvious. For the if part, if B_i is outerplanar, then the edge (c_i, c_{i-1}) can always be embedded onto an interior face of $B_i \cup \{(c_i, c_{i-1})\}$. Otherwise, there must exist an edge (x, y) with x and y lying on opposite sides of the exterior boundary divided by c_i and c_{i-1}. But then the exterior boundary and the edge (x, y) would form a $\Theta_4^{c_i, c_{i-1}}$-subdivision of B_i, contradicting Theorem 6.1. Since edge (c_i, c_{i-1}) can always be embedded onto an interior face of $B_i \cup \{(c_i, c_{i-1})\}$, the exterior boundary of $B_i \cup \{(c_i, c_{i-1})\}$ is that of B_i.

In running **Algorithm** `SP&Outerplanar`, if the algorithm aborts execution and reports that a B_k, for some $2 \leq k \leq h-1$, is not SP with source c_k and sink c_{k-1} based on Theorem 6.1 and not the discovery of a K_4-subdivision, then $B_i, k \leq i \leq h$, could still be outerplanar. We will continue to test for outerplanarity from B_k onwards but using B_i instead of $B_i \cup \{(c_i, c_{i-1})\}, k \leq i \leq h$.

When a $B_k \cup \{(c_k, c_{k-1})\}$ is verified to be non-outerplanar by a $K_{2,3}$-subdivision and not a K_4-subdivision, if the $K_{2,3}$-subdivision contains (c_k, c_{k-1}) but (c_k, c_{k-1}) is not an edge of B_k, then it is not a negative certificate for B_k. Fortunately, we can always replace (c_k, c_{k-1}) with a path in B_k to turn the $K_{2,3}$-subdivision into a negative certificate for B_k. This is accomplished as follows. First note that c_k is the root of the *dfs* tree of B_k and $(c_k \rightarrow c_{k-1})$ is its only child edge because B_k is biconnected. If c_{k-1} has no child in B_k, then $B_k \cup \{(c_k, c_{k-1})\}$ would consist of a set of parallel edges which is outerplanar, contradicting it is non-outerplanar. Let u be a child of c_{k-1} in B_k. Then the ear $P_{ear(c_{k-1} \rightarrow u)}$ contains a $c_k \rightsquigarrow c_{k-1}$ path in B_k. It is easily verified that an internal vertex of the $c_k \rightsquigarrow c_{k-1}$ path would be a cut-vertex of B_k unless there are two interlacing ears s-attached to the path or there is another $c_k \rightsquigarrow c_{k-1}$ path in B_k. In the former case, the interlacing ears give rise to a K_4-subdivision, contradicting the assumption that no K_4-subdivision was detected in $B_k \cup \{(c_k, c_{k-1})\}$. In the latter case, assume without loss of generality that the $K_{2,3}$-subdivision is extended from the edge (c_k, c_{k-1}) into the first $c_k \rightsquigarrow c_{k-1}$ path. Then by the structure of $K_{2,3}$-subdivision and the dfs tree, it is easily verified that the $K_{2,3}$-subdivision cannot be extended into the second $c_k \rightsquigarrow c_{k-1}$ path. Hence, the edge (c_k, c_{k-1}) can be replaced by the second $c_k \rightsquigarrow c_{k-1}$ path, resulting in a $K_{2,3}$-subdivision of B_k. Let e be the child edge of c_{k-1} on the second $c_k \rightsquigarrow c_{k-1}$ path. The path can be determined in $O(|V|)$ time using the back-edge $ear(e)$.

7 Recognition of generalized series-parallel graphs

Theorem 7.1. [29] *A graph is GSP if and only if its biconnected components are SP graphs.*

Theorem 7.1 shows that the problem of recognizing GSP graphs can be reduced to that of recognizing SP graphs. The idea is to decompose the graph G into its biconnected components, construct a decomposition tree for each biconnected component using **Algorithm** SP&Outerplanar, and then connect the decomposition trees using the $\texttt{SC}$ or $\texttt{DC}$ operation to form a GSP decomposition tree of G. First, we shall give a high-level description of the algorithm.

The depth-first-search-based algorithm for biconnectivity [23] is used to decompose the graph G into its biconnected components and determine its cut-vertices. For each biconnected component, let its *root vertex* be the cut-vertex through which the dfs enters the biconnected component, or the root r of the depth-first search tree if the biconnected component contains r. The biconnected components with their respective root vertex are added to a queue Q in the order they are generated. Clearly, the one containing r is entered last.

If the biconnected components (cut-vertices) form a chain as discussed in Section 6.2, a flag SP is set to *true* to indicate that; SP is set to *false*, otherwise. In the former case, let $B'_1, B'_2, \ldots, B'_h$ be the order of the biconnected components in Q, where B'_1 is the first element (note that B'_h contains r). Owing to the nature of depth-first search, B'_1 contains exactly one cut-vertex. Let B'_ℓ be the other biconnected component containing exactly one cut-vertex. Reverse the order of the biconnected components in Q starting from B'_ℓ to B'_h. Let the resulting ordered list in Q be $B_1, B_2, \ldots, B_h$. Then B_i and B_{i+1} shares a unique cut-vertex $c_i, 1 \leq i < h$. Let c_{i+1} be the root vertex of $B_i, \ell \leq i < h$.

Remove the elements from Q one at a time. Let B_i be the biconnected component removed from Q and c_i be its root vertex. Execute **Algorithm** SP&Outerplanar on B_i to generate a decomposition tree, $\mathcal{T}_{B_i}$, of B_i as follows:

(a) $i = 1$: execute **Algorithm** SP&Outerplanar on B_1, with c_1 as source and x as sink, where (c_1, x) is any edge in B_1. Attach $\mathcal{T}_{B_1}$ to c_1.

(b) $1 < i < h$: (i) $SP \equiv true$: $\mathcal{T}_{B_j}, 1 \leq j < i$, have been constructed and merged into a composition tree $\mathcal{T}$ with source c_{i-1} and sink x via the $\texttt{SC}$ operation. The tree is attached to c_{i-1}. Execute **Algorithm** SP&Outerplanar on $B_i \cup \{(c_i, c_{i-1})\}$ with c_i as source and c_{i-1} as sink. When $\mathcal{T}_{B_i}$ is constructed, merge $\mathcal{T}$ with $\mathcal{T}_{B_i}$ by $\texttt{SC}(\mathcal{T}_{B_i}, \mathcal{T})$ and attach the resulting tree to c_i. (ii) $SP \equiv false$: if K_4-$found \equiv false$, execute **Algorithm** SP&Outerplanar on $B_i \cup \{(c_i, x)\}$ with c_i as source and x as sink, where (c_i, x) is any edge in B_i. In the course of generating $\mathcal{T}_{B_i}$, whenever a cut-vertex $c'(\neq c_i)$ is encountered, owing to the properties of depth-first search and Q, the decomposition tree $\mathcal{T}_{c'}$ for all the biconnected components whose root vertex is a descendent of c' must have been constructed and attached to c'.

- c' is not the sink of B_i: $\mathcal{T}_{c'}$ is merged with $(c', p(c'))^p$ via $\mathsf{DC}((c', p(c'))^p, \mathcal{T}_{c'})$ (recall that $p(c')$ is the parent of c') to form a GSP graph with source c' and sink $p(c')$ (Figure 10(a));
- c' is the sink of B_i: $\mathcal{T}_{c'}$ is merged with $\mathcal{T}_{B_i}$ via $\mathsf{SC}(\mathcal{T}_{B_i}, \mathcal{T}_{c'})$ (Figure 10(b)).

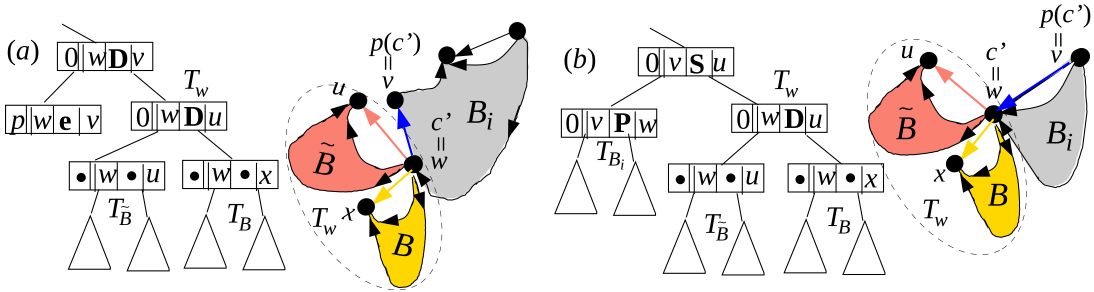

Figure 10: Connecting decomposition trees of biconnected components.

When $\mathcal{T}_{B_i}$ is constructed, if there is a decomposition tree $\mathcal{T}$ attached to c_i, merge $\mathcal{T}$ with $\mathcal{T}_{B_i}$ by $\mathsf{DC}(\mathcal{T}, \mathcal{T}_{B_i})$ and attach the resulting tree to c_i. Otherwise, just attach $\mathcal{T}_{B_i}$ to c_i.

(c) $i = h$: (i) $SP \equiv true$: execute **Algorithm** SP&Outerplanar on B_h, with s' as source and c_{h-1} as sink, where (s', c_{h-1}) is any edge in B_h. When $\mathcal{T}_{B_h}$ is constructed, merge the decomposition tree $\mathcal{T}$ attached to c_{h-1} with $\mathcal{T}_{B_h}$ via $\mathsf{SC}(\mathcal{T}_{B_h}, \mathcal{T})$. The resulting tree is a GSP decomposition tree of G. (ii) $SP \equiv false$: execute **Algorithm** SP&Outerplanar on B_h, with r as source and x as sink, where (r, x) is any edge in B_h. Merge decomposition trees attached to its cut-vertices using the SC or DC operation as explained above. When $\mathcal{T}_{B_i}$ is constructed, if r is not a cut-vertex, then $\mathcal{T}_{B_i}$ is a GSP decomposition tree of G. Otherwise, merge the decomposition tree $\mathcal{T}$ attached to r with $\mathcal{T}_{B_h}$ via $\mathsf{DC}(\mathcal{T}, \mathcal{T}_{B_h})$. If Q is empty, the resulting tree is a GSP decomposition tree of G. Otherwise, attach the resulting tree to r.

The two operations depicted in Figure 10 can be easily incorporated into **Algorithm** SP&Outerplanar by modifying the statement marked by ► as follows:

if $(v = r)$ **then if** (w is a not cut-vertex) **then** $seq := \mathsf{PC}((v, w)^p, seq)$ // $(v, w)^p \in E_s$;
else $seq := \mathsf{SC}(\mathsf{PC}((v, w)^p, seq), \mathcal{T}_w)$ // connect $\mathcal{T}_w$ to $\mathcal{T}_{B_i}(= \mathsf{PC}((v, w)^p, seq))$ via SC; (Figure 10(b))
else if (w is not a cut-vertex) **then** $seq := \mathsf{SC}(seq, (w, v)^p)$
else $seq := \mathsf{SC}(seq, \mathsf{DC}((w, v)^p, \mathcal{T}_w))$; // attach $\mathcal{T}_w$ to $(w, v)^p$ via DC first, then connect seq; (Figure 10(a))

The decomposition tree is generalized to accommodate the DC operation as follows:

- $\mathcal{T}_G$ is a binary tree with $\overline{\underline{|0||s|\mathbf{D}|t|}}$ as the root, $\mathcal{T}_{G_1}$ and $\mathcal{T}_{G_2}$ as the left and right subtrees, respectively, if $G = \mathtt{DC}(G_1, G_2)$, where s is the common source of G_1 and G_2, and t is the sink of G_1.

In executing **Algorithm** SP&Outerplanar, if a $\Theta_4^{c_i,c_{i-1}}$-subdivision is detected in B_i, SP is set to $false$, and **Algorithm** SP&Outerplanar is reinvoked on B_i with c_i as source and x (instead of c_{i-1}) as sink such that (c_i, x) is an edge in B_i. If a K_4-subdivision is detected, execution terminates with the K_4-subdivision returned as a negative certificate confirming G is not GSP, SP or outerplanar.

The following is the pseudo code of the certifying algorithm for recognizing GSP, SP and outerplanar graphs. The statements before the **repeat** loop are self-explanatory. The **repeat** loop runs **Algorithm** SP&Outerplanar on each biconnected components B_i. Within the loop, the **then** part of the first **if** statements deals with the case when it is known that G is not SP. When a $K_{2,3}$-subdivision is found in B_i, the second **if** statement checks if the $K_{2,3}$-subdivision contains the edge (c_i, c_{i-1}) that is not in B_i and replaces that edge with a path in B_i accordingly (see Section 6.3). When a K_4-subdivision is found in B_i, the **then** part of the third **if** statement checks if the K_4-subdivision is actually a Θ_4-subdivision. If no K_4-subdivision is found in B_i, the **else** part attaches the decomposition tree $\mathcal{T}_{B_i}$ to c_i according to the rules explained above. The **if** statements following the **repeat** loop generate the certificates.

Algorithm `GSP/SP/Outerplanar`

Input: The adjacency lists of a connected multigraph $G = (V, E)$.

Output:

$$\begin{cases} \mathcal{T}_{B_h} \text{ (a } GSP \text{ decomposition tree of } G), & \text{if } G \text{ is generalized series-parallel;} \\ a\ K_4\text{-}subdivision \text{ of } G, & \text{if } G \text{ is not generalized series-parallel,} \end{cases}$$

$$\text{and} \begin{cases} \mathcal{T}_{B_h} \text{ (a } SP \text{ decomposition tree of } G), & \text{if } G \text{ is series-parallel;} \\ \begin{cases} a\ K_4\text{-}subdivision \text{ of } G,\ or \\ a\ \Theta_4^{c_i,c_{i-1}}\text{-}subdivision \text{ of } G,\ or \\ \text{three cut-vertices in a biconnected component of } G,\ or \\ \text{a cut vertex in three distinct biconnected components of } G, \end{cases} & \text{if } G \text{ is not series-parallel,} \end{cases}$$

$$\text{and} \begin{cases} \text{the exterior boundary of an outerplanar embedding of } G, & \text{if } G \text{ is outerplanar;} \\ \begin{cases} a\ K_4\text{-}subdivision \text{ of } G,\ or \\ a\ K_{2,3}\text{-}subdivision \text{ of } G, \end{cases} & \text{if } G \text{ is not outerplanar.} \end{cases}$$

begin
Convert the adjacency lists of G into compact adjacency lists $L[w], \forall w \in V$;
$K_4\text{-}found := false;\ K_{2,3}\text{-}found := false;\ SP := true;$
Use Tarjan's algorithm [23] to determine the set of biconnected components $\mathcal{G}$ and the cut-vertices of G; the dfs starts from r;
Insert the biconnected components with their root vertex into a queue Q in the order the biconnected components are generated;
if (($\exists B' \in \mathcal{G}$ containing three cut-vertices) $\vee$ ($\exists$ a cut vertex belonging to three biconnected components in $\mathcal{G}$)) **then**
  $SP := false$; // G is not series-parallel
  Let $\mathcal{G} = \{B_i \mid 1 \leq i \leq h\}$; // continue to check if G is GSP or outerplanar
else order $\mathcal{G}$ as a chain $B_i, 1 \leq i \leq h$, in Q such that B_{i-1} and B_i share a cut-vertex c_{i-1}, and $c_i, 1 \leq i < h$, are distinct;
$i := 0$;
repeat // attempt to generate a GSP decomposition tree for G
  $i := i + 1$; Remove B_i and its root cut-vertex c from Q;
  if $(SP \equiv false)$ **then**

Execute **Algorithm** SP&Outerplanar on input $B_i \cup \{(c, x)\}$ with c as source, for some edge (c, x) of B_i, and $c \in \{c_i, r\}$
else // the biconnected components form a chain
if $i = 1$ **then**
Execute **Algorithm** SP&Outerplanar on input $B_1 \cup \{(c_1, x)\}$ with c_1 as source, for some edge (c_1, x) of B_1
else if $i = h$ **then**
Execute **Algorithm** SP&Outerplanar on input $B_h \cup \{(s', c_{h-1})\}$ with c_{h-1} as sink, for some edge (s', c_{h-1}) of B_h
else Execute **Algorithm** SP&Outerplanar on input $B_i \cup \{(c_i, c_{i-1})\}$ with c_i as source and c_{i-1} as sink;
if ($(K_{2,3}\text{-}found) \wedge \sim (K_4\text{-}found)$) **then** // $B_i \cup \{(c_i, c_{i-1})\}$ is not outerplanar but SP
if (the $K_{2,3}$-subdivision, $\tilde{K}_{2,3}$, returned by `Report`($K_{2,3}$) contains (c_i, c_{i-1})) **then**
if $((c_i, c_{i-1}) \notin E)$ **then** // (c_i, c_{i-1}) is not an edge in G_i
Let $\tilde{e}$ be a child-edge of c_{i-1} that is not in $\tilde{K}_{2,3}$;
Replace (c_i, c_{i-1}) with $P_{ear(\tilde{e})}$ in $\tilde{K}_{2,3}$; // generate the correct $K_{2,3}$-subdivision of B_i
if $(K_4\text{-}found)$ **then**
if $(i \notin \{1, h\} \wedge SP)$ **then** // check if it is actually a Θ_4-subdivision of B_i that is found
if (the K_4-subdivision, $\tilde{K}_4$, returned by `Report`(K_4) contains edge (c_i, c_{i-1}) which is not in B_i) **then**
Replace $\tilde{K}_4$ with $(\tilde{K}_4 \backslash \{(c_i, c_{i-1})\})$; // generate $\Theta_4^{c_i, c_{i-1}}$-subdivision
$K_4\text{-}found := false$; $SP := false$; $i := i - 1$; // Process B_i again with a sink x where (c_i, x) is an edge in B_i
else if ((c_i is a cut-vertex)$\wedge$ (there is a $\mathcal{T}$ attached to c_i)) **then** replace $\mathcal{T}$ with `DC`($\mathcal{T}, \mathcal{T}_{B_i}$)
else attached $\mathcal{T}_{B_i}$ to c_i;
until $((i = h) \vee K_4\text{-}found)$;
if $(\sim (K_{2,3}\text{-}found \vee K_4\text{-}found))$ **then** connect the exterior boundary of $B_i, 1 \le i \le h$, to form the exterior boundary of G;

if $(K_4\text{-}found)$) **then output**(the K_4-subdivision); // G is not GSP, SP, and OP
else if $(SP \wedge \sim K_{2,3}\text{-}found)$ **then output**($\mathcal{T}_{B_h}$, the exterior boundary of G); **stop;** // G is GSP, SP, and OP
if $(SP \wedge K_{2,3}\text{-}found)$ **then output**($\mathcal{T}_{B_h}$; the $K_{2,3}$-subdivision); **stop;** // G is GSP, SP, and not OP
if $(\sim SP \wedge \sim K_{2,3}\text{-}found)$ **then** // G is GSP, OP, and not SP
output($\mathcal{T}_{B_h}$, the exterior boundary of G; $\begin{cases} \text{the } \Theta_4\text{-subdivision, or} \\ \text{three cut-vertices in a biconnected component, or} \\ \text{a cut vertex in three distinct biconnected components,} \end{cases}$); **stop;**
if $(\sim SP \wedge K_{2,3}\text{-}found)$ **then** // G is GSP, not SP and not OP
output($\mathcal{T}_{B_h}$; the $K_{2,3}$-subdivision, $\begin{cases} \text{the } \Theta_4\text{-subdivision, or} \\ \text{three cut-vertices in a biconnected component, or} \\ \text{a cut vertex in three distinct biconnected components,} \end{cases}$); **stop;**
end.

Algorithm SP&Outerplanar has to be slightly modified as follows: insert the instruction '$K_4\text{-}found := true$' in between each occurrence of `Report`(K_4) and **stop**; remove $K_{2,3}\text{-}found := false$ so that $K_{2,3}$-$found$ will not be reset to $false$ after a $K_{2,3}$-subdivision is found.

Theorem 7.2. ***Algorithm*** *GSP/SP/Outerplanar generates the certificates for G in $O(|V| + |E|)$ time.*

Proof. The correctness of generating the negative certificates indicating G is not SP before the **repeat** loop and of generating the queue Q so that B_i and B_{i+1} share a unique common cut-vertex $c_i, 1 \le i < h$, if $SP \equiv true$, are obvious. The correctness of the **repeat** loop generating a decomposition tree of G if G is SP or GSP, and the negative certificates if G is not GSP, SP, or outerplanar is easily verified by induction on i based on the correctness of **Algorithm** SP&Outerplanar and Section 6. The correctness of generating the output by the last two **if** statements are also obvious.

Converting the adjacency list of G into compact adjacency lists takes $O(|V| + |E|)$ time. Since the size of the compact adjacency-lists structure is bounded by $O(|V|)$, decomposing G into the biconnected components and building Q take $O(|V|)$ time. Generating the two types of negative certificates for SP graphs before the **repeat** loop clearly takes $O(|V|)$ time. The **repeat** loop takes $O(|E_i|)$ time per iteration based on Theorems 4.3 and 4.6, Sections 4.3.1, 4.3.2 and 6, where E_i is the edge set of B_i. The **repeat** loop thus takes $\sum_{i=1}^{h} O(|E_i|) = O(|V|)$ time. The last two **if** statements clearly take $O(|V|)$ time. $\square$

Authentication of the GSP decomposition tree is same as that for SP graphs except that on encountering an internal node $|\overline{\underline{0\|s|\mathbf{D}|t}}|$, let $s_- := s_- - 1$.

For each biconnected component B, let c be its source and x be its sink. By Section 5.1, after the traversal backtracked to the root node $|\overline{\underline{0\|c|\chi|x}}|$ of $\mathcal{T}_B$, $c_- = x_+ = 1, c_+ = x_- = 0$ and $v_- = v_+ = 0, v \in V_B \setminus \{c, x\}$. Let there be $h(> 1)$ biconnected components with c as source and B be the first one encountered (the case where $h = 1$ is similar but simpler). Their decomposition trees are connected by a chain of $h - 1$ $|\overline{\underline{0\|c|\mathbf{D}|x}}|$ nodes. At each such node, since $c_- = 1$ at each child node and c_- is decreased by 1 at the node, $c_- = 1$ when the traversal backtracks from that node. If $c = r$ (the root of the dfs tree), the traversal terminates at the $|\overline{\underline{0\|c|\mathbf{D}|x}}|$ node encountered last and $r_- = c_- = 1$. Clearly, $r_+ = c_+ = 0$. If $c \neq r$, (a) if c is not a sink, the parent node of the $|\overline{\underline{0\|c|\mathbf{D}|x}}|$ node encountered last is $|\overline{\underline{0\|c|\mathbf{D}|p(c)}}|$ and the sibling is $|\overline{\underline{\ell\|c|\mathbf{e}|p(c)}}|$ (Figure 10(a)). Again, as $c_- = 1$ at $|\overline{\underline{0\|c|\mathbf{D}|x}}|$ and $|\overline{\underline{\ell\|c|\mathbf{e}|p(c)}}|$ and c_- is decreased by 1 at $|\overline{\underline{0\|c|\mathbf{D}|p(c)}}|$, $c_- = 1$ when the traversal backtracks from $|\overline{\underline{0\|c|\mathbf{D}|p(c)}}|$. Since c is not a sink, $|\overline{\underline{0\|c|\mathbf{D}|p(c)}}|$ must have a parent node $|\overline{\underline{0\|y|\mathbf{S}|p(c)}}|$ and a sibling $|\overline{\underline{k\|y|\chi|c}}|$. When the traversal backtracks to $|\overline{\underline{0\|y|\mathbf{S}|p(c)}}|$, $c_- = c_+ = 0$ or the certificates is rejected. (b) if c is a sink, let the corresponding source be $\tilde{c}(= p(c))$. Then, there exists a node $|\overline{\underline{0\|\tilde{c}|\mathbf{S}|x}}|$ with $|\overline{\underline{0\|\tilde{c}|\mathbf{P}|c}}|$ as the left child and $|\overline{\underline{0\|c|\chi|x}}|$ as the right child (Figure 10(b)). After the traversal backtracked to node $|\overline{\underline{0\|\tilde{c}|\mathbf{S}|x}}|$, $c_- = c_+ = 0$ or the certificates is rejected. For each sink x that is not a cut-vertex, $x_+ = 1$ and $x_- = 0$ remain unchanged. Hence, when the traversal terminates at the root node $|\overline{\underline{0\|r|\chi|t}}|$ of the decomposition tree of G, if $(r_- = 1 \wedge v_- = 0, v \in V \setminus \{r\})$ and $\left(v_+ = \begin{cases} 1, & v \text{ is a sink and not a cut-vertex;} \\ 0, & v \text{ is a sink and a cut-vertex.} \end{cases} \wedge v_+ = 0, v \text{ is not a sink}\right)$, then precede to check if $\tilde{L}[v], v \in V$ are adjacent lists of G as in Section 5.1; reject the certificate, otherwise.

The authentication of $K_{2,3}$-subdivision, K_4-subdivision, Θ_4-subdivision are same as or similar to before. The authentication of the negative certificates indicating G has three connected components sharing a common cut-vertex or a connected component containing three cut-vertices can clearly be done in $O(|V|)$ time.

8 Conclusion

We presented the first $O(|V| + |E|)$-time certifying algorithm for determining if a multigraph $G = (V, E)$ is generalized series-parallel and, if it is, to which subclass of generalized series-parallel graphs G belongs. The algorithm only makes one pass over G after a preprocessing step. It also generates certificates for verifying the correctness of the output. We also presented simple authentication algorithms for verifying the certificates.

Acknowledgement

This research was supported by HK GRF grant HKU7114/13E and HKU Outstanding Researcher Award, HK GRF grant HKU7164/12E, U of Windsor GRF grant #815160, and NSFC(No. 61433012, U1435215) and Shenzhen Research Grant (KQJSCX20180330170311901, JCYJ20180305180840138).

The authors thank the first referee for pointing out an error in the first draft.

References

[1] Brehaut W.M., "An efficient outerplanarity algorithm," Proc. 8th Southeastern Conference on Combinatorics, Graph Theory and Computing, Baton Rouge, Louisiana, Feb 1977, 99-113.

[2] Corneil D., Dalton B., and Habib M., "LDFS based certifying algorithm for the Minimum Path Cover problem on cocomparability graphs," SIAM J. Comput., vol.42(3), 792-807 (2013)

[3] Duffin R.J., Topology of series parallel networks, J. Math. Anal. & Appl. vol.10, 303-318 (1965)

[4] Elmasry A., Mehlhorn K., Schmidt J.M., An $O(n + m)$ Certifying Triconnnectivity Algorithm for Hamiltonian Graphs, Algorithmica 62(3-4): 754-766 (2012).

[5] Eppstein D., Parallel recognition of series-parallel graphs, Information and Computation, vol.98, 41-55 (1992).

[6] Francis M.C., Hell P., Stacho J., Forbidden structure characterization of circular-arc graphs and a certifying recognition algorithm, SODA'15, San Diego, CA, Jan 4-6, 2015, 1708-1727.

[7] Fussell D., Ramachandran V., Thurimella R., Finding triconnected components by local replacement, SIAM Journal on Computing 22(3), 587-616, 1993.

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

[9] Hare E., Hedetniemi S., Larkar R., Peters K., Wimer T., "Linear-time computability of combinatorial problems on generalized series-parallel graphs," Discrete algorithms and Complexity, Academic Press, 437-455, (1987).

[10] Hopcroft J. and Tarjan R.E., "Efficient planarity testing," J. ACM vol.21(4), 549-568 (1974).

[11] Kanevsky A., Ramachandran V., Improved algorithms for graph four-connectivity, Journal of Computer and System Sciences 42, 288-306, 1991.

[12] Korenblit M., Levit V., On algebaric expressions of series-paralle and Fibonacci graphs, DMTCS 2003, LNCS 2731, 215-224 (2003)

[13] Kratsch D., McConnell R., Mehlhorn K., Spinrad J., Certifying algorithms for recognizing interval graphs and permutation graphs, SIAM Journal on Computing 36(2), 326-353, 2006.

[14] McConnell R.M., Mehlhorn K., Naher S., and Schweitzer P., Certifying algorithms, Computer Science Review vol.5, 119-161 (2011).

[15] 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 (MFCS98), 1998, 8493.

[16] Mehlhorn K., Neumann A., and Schmidt J.M., Certifying 3-edge-connectivity, WG2013, LNCS vol.8165, 358-369 (2013).

[17] Mitchell S.L., Linear algorithms to recognize outerplanar and maximal outerplanar graphs, Information Processing Letters, vol.9(5), 229-232 (1979).

[18] Proskurowski A., Sysło M., “Efficient vertex-and edge-coloring of outerplanar graphs,” SIAM Journal on Algebraic and Discrete Methods, vol.7 131136, (1986).

[19] Reif J.H., “Depth-first search is inherently sequential,” Information Processing Letters, vol.20(5), 229-234 (1985).

[20] Schoenmakers B., A new algorithm for the recognition of series parallel graphs, CWI Report CS-R9504, January (1995).

[21] Sysło M.M. and Iri M., “Efficient outerplanarity testing,” Annales Societatis Mathematicae Polonae Series IV: Fundamenta Informaticae II, 261-275 (1979).

[22] Takamizawa K., Nishizeki T., Saito N., “Linear-time computability of combinatorial problems on series-parallel graphs.” Journal of the ACM. 29(3), 623641 (1982).

[23] Tarjan R.E., Depth-first search and linear graph algorithms, SIAM J. Comput. 1(2), 146-160 (1972)

[24] Tsin Y.H., Recognizing and embedding outerplanar distributed computer networks, CyberC 2011, IEEE, Beijing, China, Oct 10-12, 2011, 212-219.

[25] Tsin Y.H., A simple certifying algorithm for 3-edge-connectivity, CoRR abs/2002.04727 (Feb 11, 2020).

[26] Valdes, J., Tarjan, R.E. and Lawler, E., The recognition of series parallel digraphs, SIAM J. Comput 11(2), 298-313 (1982)

[27] Wiegers, M., “Recognizing outerplanar graphs in linear time,” Proc. WG86, Bernried, Germany, LNCS vol. 246, Springer Verlag, 165-176 (1987).

[28] Whitney H., *Non-separable and planar graphs*, Transactions of the American Mathematical Society vol.34, 339-362 (1932)

[29] Wimer T.V., Hedetniemi S.T., K-terminal recursive families of graphs, Congressus Numerantium 63, 161-176 (1988).