Indexing All Rooted Subgraphs of a Rooted Graph

Tomoki IMADA¹, Nonmember and Hiroshi NAGAMOCHI¹(a), Member

SUMMARY Let G be a connected graph in which we designate a vertex or a block (a biconnected component) as the center of G. For each cut-vertex v, let Gv be the connected subgraph induced from G by v and the vertices that will be separated from the center by removal of v, where v is designated as the root of Gv. We consider the set R of all such rooted subgraphs in G, and assign an integer, called an index, to each of the subgraphs so that two rooted subgraphs in R receive the same indices if and only if they are isomorphic under the constraint that their roots correspond each other. In this paper, assuming a procedure for computing a signature of each graph in a class G of biconnected graphs, we present a framework for computing indices to all rooted subgraphs of a graph G with a center which is composed of biconnected components from G. With this framework, we can find indices to all rooted subgraphs of a outerplanar graph with a center in linear time and space.

key words: graph isomorphism, index, rooted graphs, outerplanar graphs, signature

1. Introduction

Graph isomorphism is one of the most fundamental and well known problems in the graph theory, which has many applications. In spite of efforts of many researchers for a long time, many problems on graph isomorphism remain to be open. No essential necessary and sufficient conditions for given two graphs G1 and G2 to be isomorphic have been found. There has been no polynomial time algorithm for testing whether G1 and G2 are isomorphic or not. Under these circumstances, graph isomorphism attracts many researchers even now.

However, polynomial time algorithms were proposed for particular classes of graphs such as graphs of bounded degrees [13], partial k-trees for a constant k [2], and so on. For some classes of graphs, linear-time algorithms were proposed. For example, a linear-time algorithm for testing isomorphism of planar graphs was reported by Hopcraft and Wong [5]. They introduced a signature of a given planar graph which can be computed in linear time. A signature for a class G of graphs is a function from a graph in G to a string of integers such that two graphs in G have the same signatures if and only if they are isomorphic. In particular, for the class of outerplanar graphs, Manning and Atallah [14] proposed a signature which can be obtained by a simple procedure.

In this paper, we treat a connected graph G with vertex and edge labelings, where some two vertices (edges) may receive the same label. Suppose that a cut-vertex or block (a biconnected component) in G is designated as the center of G. In the example of a graph G in Fig. 1, the block B* indicated by the dashed circle is designated as the center of G. For each cut-vertex v, let Gv be the connected subgraph induced from G by v and the vertices that will be separated from the center by removal of v, where v is designated as the root of Gv. For the example in Fig. 1, the subgraph Gv for the cut-vertex v is indicated by the circle marked with Gv. We consider the set R of all such rooted subgraphs Gv of G. Our aim is to assign an integer, called an index, to each of the subgraphs so that two rooted subgraphs in R receive the same indices if and only if they admit an isomorphism with respect to the vertex and edge labelings such that their roots correspond each other. For example, assuming that all vertices (resp., edges) have the same label in the graph G in Fig. 1, each of the 14 cut-vertices in G is indexed by an integer from 1 to 7, and the set R contains seven non-isomorphic rooted subgraphs. Note that an index for a class of graphs is different from a signature for the class in the sense that indices for two rooted graphs G1 and G2 are defined independently and two non-isomorphic rooted subgraphs G' i of G i (i = 1, 2) may happen to have the same integers as their indices. For a rooted tree G, Dinitz et al. [3] and other researchers have presented polynomial time algorithms for testing whether two rooted trees are isomorphic or not.
showed that indices of all the subtrees rooted at cut-vertices can be computed in linear time and space without using any integers greater than the number of vertices during the computation process.

In this paper, we extend the result of Dinitz et al. to a wider class of connected graphs whose blocks belong to a class $\mathcal{G}$ of biconnected graphs such that a signature of each graph in $\mathcal{G}$ is available. For this, we introduce indices for a different type of subgraphs in a graph with a center in the following way. For each block $B$ in a graph $G$, let $r_B$ the vertex in $B$ closest to the center of $G$ and let $G_B$ be the connected subgraph induced from $G$ by $v$ and the vertices that will be separated from the center by removal of the edges in $B$, where $r_B$ is designated as the root of $G_B$. In Fig. 1, the graph $G_B$ for the block $B$ is indicated by the circle marked with $G_B$ and $r_B = v$. Let $\mathcal{R}_B$ denote the set of all such rooted subgraphs $G_B$ in $G$. For a graph $G$ in a class of connected graphs, we compute indices to all rooted subgraphs in $\mathcal{R}_B$ and $\mathcal{R}$ alternatively in a nondecreasing order of the size of $G_B$ and $G_B$. The key idea is that we can define a “signature” $S(v)$ (a string of integers) of $G$, from the indices of all rooted subgraphs $G_B$ such that $v = r_B$ and that signatures will be mapped into indices of rooted graphs $G_{\mathcal{R}}$ by computing a “ranking” over the set of signatures.

Note that if arbitrarily large integers are allowed to be used during a computation process, then the signatures of any number of rooted subgraphs could be represented by a huge but simple integer, which would reduce the space complexity meaninglessly. Hence we also evaluate upper and lower bounds of integers used for storing indices during the computation process of an algorithm.

For a class $\mathcal{G}$ of biconnected graphs, let $G^*$ denote the class of connected graphs whose blocks belong to $\mathcal{G}$. In this paper, we present a framework for computing indices of all rooted subgraphs $\mathcal{R}_B \cup \mathcal{R}_B$ in a graph $G \in G^*$ based on a procedure for computing a signature of each graph with vertex and edge labelings in $G$. For a given graph $G = (V, E)$, any algorithm provided by our framework runs in $O(|V| + |E|)$ time and space except for the time and space complexities for computing signatures, and uses only integers from 0 to $\max(2|V| - 1, |V| + 1, |E| + 1)$ except for the process for computing signatures of blocks.

As an application of our result, we consider rooted outerplanar graphs with vertex and edge labelings. Recently computational enumeration of chemical compounds has been studied extensively [4], [7]–[9], [11], [12]. A chemical compound is modeled as a graph with vertex and edge labelings such that each vertex label represents a kind of atom, such as a carbon, nitrogen, and so on, and each edge label represents the multiplicity of a bond between two atoms. It is known that 94.3% of chemical compounds in the NCI chemical database are outerplanar graphs [6]. Thus studying the structure of outerplanar graphs with vertex and edge labelings is important. In particular, for enumerating stereoisomers of a given chemical compound [7]–[9], indexing all its rooted subgraphs plays a crucial role at the beginning of the computation process because two of its subgraphs cannot be the same stereoisomer without considering their three dimensional structures if their graph structures are not isomorphic. Based on the signature given by Manning and Atallah for outerplanar graphs with no vertex or edge labelings [14], we introduce a signature of rooted blocks for outerplanar graphs with vertex and edge labelings. By applying the signature to our framework, we can compute indices of all the subgraphs rooted at vertices in a given outerplanar labeled graph in linear time and space, where the computation process uses only integers from 0 to $\max(2|V| - 1, |V| + 1)$.

The rest of this paper is organized as follows. Section 2 introduces some definitions and notations, and Sect. 3 reviews a hierarchical structure of the block-cut-vertex tree of a connected graph. Section 4 gives our framework for computing indices on a rooted graph and analyzes its time and space complexities. Section 5 shows how to compute signatures of outerplanar rooted blocks with vertex and edge labelings. Section 6 makes some concluding remarks.

2. Preliminary

For two integers $i$ and $j$ such that $i \leq j$, let $[i, j]$ denote the set $\{i, i + 1, i + 2, \ldots, j\}$ of integers.

In this paper, a labeled graph stands for a simple undirected graph $G = (V, E)$ with a vertex labeling $\psi$ from its vertex set $V$ to a vertex label set $\Lambda^V$ and an edge labeling $\hat{\Lambda}$ from its edge set $E$ to an edge label set $\Lambda^E$, where we assume without loss of generality that the label of each vertex is given as a positive integer in $[1, |\Lambda^V|]$ and that of each edge is given as a positive integer in $[1, |\Lambda^E|]$. The label $\hat{\Lambda}(e)$ of an edge $e = (u, v)$ is also denoted by $\hat{\Lambda}(u, v)$. By definition, it holds that $|\Lambda^V| \leq |V|$ and $|\Lambda^E| \leq |E|$. The sets of vertices and edges of a graph $G$ are also denoted by $V(G)$ and $E(G)$, respectively. A rooted graph is a graph in which a vertex is designated as the root.

Two labeled graphs $G_1$ and $G_2$ are isomorphic if there is a bijection $\psi : V(G_1) \rightarrow V(G_2)$ with $\psi(v) = (\psi(v), v \in V(G_1))$ such that $(\psi(e)) \in E(G_2)$ contains an edge $e = (u, v)$ if and only if $E(G_2)$ contains an edge $e' = (\psi(u), \psi(v))$ with $\hat{\Lambda}(e') = \hat{\Lambda}(e)$. Such a bijection is called an isomorphism between $G_1$ and $G_2$. Two subgraphs $G$ and $G'$ with roots $r \in V(G)$ and $r' \in V(G')$ are rooted-isomorphic, denoted by $G \cong G'$, if they admit an isomorphism $\psi$ with $\psi(r) = r'$. Indices for a set $R$ of rooted graphs are defined to be non-negative integers $\text{id}(G), G \in R$ such that

$$\text{id}(G') \Rightarrow G \equiv G'.$$

In this paper, we are given a connected labeled graph $G$. A cut-vertex of $G$ is a vertex whose removal results in more than one connected graphs. A graph is biconnected if it is connected and has no cut-vertex. The blocks (or biconnected components) of a graph are its maximal biconnected subgraphs. A block is also called a bridge if it is composed of two vertices and an edge joining them. The set of blocks in a graph $G$ is denoted by $B(G)$.

Let $G$ have a center which is designated by a cut-vertex
Theorem 2: Let \( G \), \( a \), and \( b \) be a set of strings whose entries are integers in the range from 0 to \( m - 1 \), and \( \ell_i \) be the length of \( A_i = (a_{i1}, a_{i2}, \ldots, a_{i\ell_i}) \). Then the lexicographical sort of \( A_1, A_2, \ldots, A_n \) can be done in \( O(m + \ell_{\text{total}}) \) time, where \( \ell_{\text{total}} = \sum_{i=1}^{n} \ell_i \).

Their algorithm for sorting the set of strings is a modification of radix sort. Though they do not write explicitly about the space complexity, it is easy to see that their algorithm runs in \( O(m + \ell_{\text{total}}) \) space.

In this paper, we assume that all the blocks of a given graph \( G \) belong to a class \( \mathcal{G} \) of biconnected graphs and use the following computability of the signature on the class \( \mathcal{G} \).

Definition 3: For a class \( \mathcal{G} \) of biconnected graphs and any positive integers \( a \) and \( b \), let \( (\mathcal{G}, a, b) \) be the set of rooted blocks with its vertex label set \( [1, a] \) and edge label set \( [1, b] \) which belong to a graph class \( \mathcal{G} \). Let a signature of a given set \( (G, a, b) \) be given as a function \( \sigma \) that maps \( B \in (\mathcal{G}, a, b) \) to a string \( \sigma(B) \). The signature \( \sigma \) is defined to be computable by integer functions \( (L, M, X, S) \) if it satisfies the following three conditions for any block \( B \in (\mathcal{G}, a, b) \):

(i) The length \( |\sigma(B)| \) of the signature is \( O(L(B)) \).
(ii) Each element of the signature \( \sigma(B) \) is an integer \( i \in [0, \max\{a + 1, b + 1, M(B)\}] \).
(iii) The signature \( \sigma(B) \) can be constructed in \( O(X(B)) \) time and in \( O(S(B)) \) space, provided that any integer in \( [0, \max\{a + 1, b + 1, M(B)\}] \) can be stored in unit space and each of addition, subtraction, multiplication, and division over such integers can be executed in unit time.

In this paper, we fix a given graph \( G = (V, E) \), denote by \( M_{\text{max}} \) the maximum integer \( M(B) \) for all \( B \in \mathcal{B}(G) \), and assume that integers from 0 to at most \( \max\{|V| + 1, |E| + 1, M_{\text{max}}, 2|V| - 1\} \) can be stored in unit space and each of addition, subtraction, multiplication, and division over such integers can be executed in unit time.

3. Hierarchy in Rooted BCL-Trees

We try to determine indices of the subgraphs \( G_a \) and \( G_b \) in a nondecreasing order of the size of \( G_a \) and \( G_b \). To formulate the method, we use block-cut-vertex tree (BC-tree, for short) of a connected graph \( G \), a representation for the relationship among cut-vertices and blocks in \( G \). The BC-tree \( \mathcal{T}(G) \) of \( G \) is a bipartite tree in a set of B-nodes and a set of C-nodes such that (i) there is exactly one B-node (resp., C-node) for each block (resp., cut-vertex) of \( G \), and (ii) there is an edge between a B-node and a C-node if and only if the block corresponding to the B-node contains the cut-vertex corresponding to the C-node in \( G \). Figure 2(a) illustrates the BC-tree of the graph \( G \) in Fig. 1. Throughout the paper, “nodes” are used for BC-trees \( \mathcal{T}(G) \) (and its extension introduced later) whereas “vertices” are used for labeled graphs \( G \). For notational convenience, the node corresponding to a block \( B \) (resp., a vertex \( v \)) is also referred to as \( B \) (resp., \( v \)). The BC-tree of a given connected graph \( G \) can be constructed in \( O(|E(G)|) \) time [1].

The center of \( G \) corresponds to exactly one node of the BC-tree of \( G \), which we call the root-node of \( \mathcal{T}(G) \). The presence of the root-node in a tree introduces the standard parent-child relationship among the nodes in the tree. For each non-root node \( x \) in \( \mathcal{T}(G) \), let \( p(x) \) denote the parent of \( x \). A non-center vertex \( v \) in \( G \) is called leaf-vertex if \( v \) is not

---

1.In this paper, the center can be any cut-vertex or any block, and plays as the root of the given graph \( G \). We use “center” only for the entire graph \( G \) to distinguish it from the root of a proper subgraph \( G_a \) or \( G_b \) which is always a vertex. If the center is chosen as a special vertex/block which is uniquely determined by the entire structure of \( G \), then two rooted graphs are isomorphic under the constraint that their roots correspond each other if and only if they are isomorphic without such a constraint.
a cut-vertex. In Fig. 2(a), each vertex with numbered by 0 in Fig. 1 is a leaf-vertex. A non-center block which contains exactly one cut-vertex in $G$ is called a leaf-block. Each leaf-node of the BC-tree in Fig. 2(a) corresponds to a leaf-block in Fig. 1.

For notational convenience in describing our method for computing indices of rooted subgraphs, we slightly extend the notion of BC-trees as follows. The block-cut-vertex-leaf-vertex tree (BCL-tree, for short) $T(G) = (N, E)$ of a connected graph $G$ with a center is defined to be the rooted tree obtained from the BC-tree $\tilde{T}(G)$ of $G$ by adding new vertices of degree 1, called L-nodes as follows: (i) For each leaf-vertex $v \in V$, there is exactly one L-node $u_v$; and (ii) the edge incident to an L-node $u_v$ is incident to the B-node $B$ which contains the corresponding leaf-vertex $v$. Note that there are no edges between a C-node and an L-node. See Fig. 2(b) for an example. By definition, all leaves of a BCL-tree are L-nodes.

We define the height of a node $u \in V(T(G))$ to be the length of the path from $u$ to the furthest leaf-node of its subtree rooted at $u$. See Fig. 2(b) for an example. Let $h(T(G))$ denote the height of the root-node of $T(G)$. Let $N(i)$ be the set of nodes of height $i$ in the BCL-tree. We can compute heights of all nodes $u \in V(T(G))$ and construct $N(i)$ for all $i \in [0, h(T(G))]$ from the leaf-nodes toward the root-node.

For each even integer $j \in [0, h(T(G))]$, let $C(j) \subseteq V(G)$ be the set of cut/leaf-vertices corresponding to the nodes in $N(j)$.

For each odd integer $i \in [1, h(T(G))]$, let $B(i) \subseteq B(G)$ be the set of blocks corresponding to B-nodes in $N(i)$, and we define root($i$) to be the list of the root-vertices $r_B$ of all blocks $B \in B(i)$ sorted by their labels $l(r_B) \in \Lambda$ in a nondecreasing order, and let edge($i$) be the list of the edges $e$ in all blocks $B \in B(i)$ sorted by their labels $l(e) \in \Lambda^2$ in a nondecreasing order. Note that root($i$) (resp., edge($i$)) may contain more than one root-vertex (resp., edge) with the same labels.

Each non-leaf-vertex belongs to at least one of the lists root($i$), $i \in [1, h(T(G))]$ and each edge belongs to exactly one of the lists edge($i$), $i \in [1, h(T(G))]$. If the height of each block is computed, we can easily determine the lists that contain a specified vertex $v \in V$ and the list that contains a specified edge $e \in E$.

For a node $x$ in $T(G)$, let $G_x$ denote the subgraph $G$ if $x$ is a vertex $v$, or $G_B$ if $x$ is a B-node $B$. Note that the rooted subgraph $G_x$ with a node $x \in N(i)$ is not rooted-isomorphic to the rooted subgraph $G_y$ with any node $y \in N(j)$ if $i \neq j$ except for the following case.

**Lemma 4:** [10] For two nonnegative integers $i > j$, $G_x$ with a node $x \in N(i)$ is rooted-isomorphic to $G_y$ with a node $y \in N(j)$ if and only if $j = i - 1$, $x$ is a C-node, and $x$ has a unique child $y' \in N(i - 1)$ in the BCL-tree such that $G_x \cong G_{y'}$ holds.

Let $\overline{N}(i) = \{x \in N(i) \mid x$ is a C-node which has exactly one child in the BCL-tree$, where $\overline{N}(i) = \emptyset$ if $i$ is odd or $i = 0$. Let $\mathcal{R}(i)$ denote the maximal set of subgraphs $G_x$ with nodes $x \in N(i) \setminus \overline{N}(i)$ such that no two subgraphs in $\mathcal{R}(i)$ are rooted-isomorphic. By Lemma 4, $G_x$ for each node $x \in \mathcal{R}(i)$ is not rooted-isomorphic to $G_y$ for any node $y \in \mathcal{R}(j)$, $j \neq i$. Hence the entire set of rooted-nonisomorphic subgraphs $G_x$ with $x \in N$ is partitioned into $\mathcal{R}(i)$, $i = 0, 1, \ldots, h(T(G))$.

**Lemma 5:** [10] For a connected graph $G = (V, E)$, the number of blocks in $G$ is at most $|V| - 1$.

By Lemma 5, the number of subgraphs rooted at vertices and subgraphs trimmed at blocks is at most $|V| + (|V| - 1) = 2|V| - 1$. Hence the total number of subgraphs in $\cup_{B \in B(G)} \mathcal{R}(i)$ is at most $2|V| - 1$.

4. Indices of Rooted Subgraphs

4.1 Indices by Rankings

Recall that every two rooted subgraphs in $\cup_{B \in B(G)} \mathcal{R}(i)$ are not rooted-isomorphic, where $|\cup_{B \in B(G)} \mathcal{R}(i)| < 2|V|$. Thus, indexing of these subgraphs is a bijection between
For each set of indices \(i \in [1,|\mathcal{N}(i)|] \cup [0,|\mathcal{R}(i)|]\). We choose our indices according to the heights of nodes in \(T(G)\). Thus the maximum index of a subgraph in \(\mathcal{R}(i)\) is given by

\[
id_{\text{max}}(i) = \sum_{0 \leq j \leq i} |\mathcal{R}(j)|, \quad i \in [0,h(T(G)) - 1].
\]

For each \(i \in [0,h(T(G))]\), we set the index \(id(G_x)\) of any node \(x \in \mathcal{N}(i)\) with its unique child \(y \in \mathcal{N}(i-1)\) by

\[
id(G_x) := \text{id}(G_y).
\]

and we choose each integer in \([id_{\text{max}}(i) - 1, id_{\text{max}}(i) + |\mathcal{R}(i)|]\) as the index \(id(G_x)\) of some node \(x \in \mathcal{N}(i) \setminus \mathcal{N}(i)\). For this, we will represent all subgraphs \(G_x, x \in \mathcal{N}(i) \setminus \mathcal{N}(i)\) as signatures \(S(x)\) (strings of integers) such that \(S(x) = S(y)\) if and only if \(G_x\) and \(G_y\) are root-omorphically. We denote by \(S(i)\) the set of these signatures \(S(x), x \in \mathcal{N}(i) \setminus \mathcal{N}(i)\). Then we will find a ranking \(\rho^{(i)}(S(x))\) over \(S(i)\), based on which the index \(id(G_x)\) for each \(x \in \mathcal{N}(i) \setminus \mathcal{N}(i)\) is given as

\[
id(G_x) := id_{\text{max}}(i) + \rho^{(i)}(S(x)).
\]

Thus the task for determining the indices by (2) is to compute the rankings \(\rho^{(i)}\) for \(i = 1,2,\ldots,h(T(G))\) in this order.

### 4.2 Sketch of Algorithm

Our algorithm determines the indices \(id(G_x), x \in \mathcal{N}(i)\) for each height \(i = 0,1,\ldots,h(T(G))\) in this order. For this, we compute a list \(\mathcal{N}_{\text{child}}(j) (j \in [0,h(T(G))]\) of the children of all nodes \(x \in \mathcal{N}(j)\) sorted in a nondecreasing order by the indices \(id(G_x)\). Note that \(\mathcal{N}_{\text{child}}(j)\) may contain two nodes \(x\) and \(y\) such that \(id(G_x) = id(G_y)\). Let \(\text{Id}(\mathcal{N}_{\text{child}})\) denote the set of indices \(id(G_x)\) for all nodes \(x \in \mathcal{N}(i)\) (note that any two indices in \(\text{Id}(\mathcal{N}_{\text{child}})\) are distinct).

**Algorithm** \(\text{Index}(G)\)

**Input:** A labeled graph \(G = (V,E)\) and its center.

**Output:** Indices of all the subgraphs \(G_x\) rooted at vertices \(v \in V\).

1. Construct the BCL-tree \(T(G)\) of \(G\), the sets \(\mathcal{N}(i), i = 0, 1,\ldots,h(T(G))\) and the lists root(i) and edge(i) for all odd \(i \in [1,h(T(G))]\).
2. Initialize the lists \(\mathcal{N}_{\text{child}}(j)\) to be the empty lists for all \(j = 0, 1,\ldots,h(T(G))\).
3. For \(i = 0, 1,\ldots,h(T(G))\) do
   - Compute a ranking \(\rho^{(i)}_{\text{child}}\) over the set of the indices \(id(G_x)\) of all nodes \(x \in \mathcal{N}_{\text{child}}(i)\).
   - Construct a signature \(S(x)\) (a string of integers) for each node \(x \in \mathcal{N}(i) \setminus \mathcal{N}(i)\) using the ranking \(\rho^{(i)}_{\text{child}}\).
   - Compute a ranking \(\rho^{(i)}\) over the set \(S(i)\) of the signatures \(S(x), x \in \mathcal{N}(i) \setminus \mathcal{N}(i)\).
   - Set the index \(G_x\) of each \(x \in \mathcal{N}(i)\) by (1) and (2); i.e.,
     \[
id(G_x) := \begin{cases} id(G_y), & \text{if } x \text{ is a C-node with exactly one child } y, \\ id_{\text{max}}(i) + \rho^{(i)}(S(x)), & \text{otherwise}; \end{cases}
\]
   - Sort all the nodes \(x \in \mathcal{N}(i)\) in a nondecreasing order by the indices \(id(G_x)\).
   - Scan the sorted list and append each node \(x \in \mathcal{N}(i)\) at the end of the list \(\mathcal{N}_{\text{child}}(k)\) with the index \(k \geq i + 1\) such that the parent \(p(x)\) of \(x\) belongs to \(\mathcal{N}(k)\).

**Lemma 6:** When algorithm \(\text{Index}(G)\) starts the iteration for an integer \(i \in [0,h(T(G))]\), the list \(\mathcal{N}_{\text{child}}(i)\) has been computed correctly.

**Proof:** By definition, \(\mathcal{N}_{\text{child}}(0)\) is an empty list. Let \(i \geq 1\). The nodes in the list \(\mathcal{N}_{\text{child}}(i)\) must have been added to the list during the iterations for \(j = 0, 1,\ldots,i-1\). By (1) and (2) the subgraph \(G_x\) for any node \(x \in \mathcal{N}(j)\) never gets an index smaller than that of any node \(y \in \mathcal{N}(j')\) with \(j' < j\), and the nodes \(x \in \mathcal{N}(j)\) are scanned in a nondecreasing order sorted by the indices \(id(G_x)\). Hence the list \(\mathcal{N}_{\text{child}}(i)\) resulting from appending these nodes at the end of the list stores nodes \(y\) in a nondecreasing order of the indices \(id(G_y)\).

The rest of this section is organized as follows. Section 4.3 describes how to construct the lists \(\text{root}(i)\) and \(\text{edge}(i)\) for all odd \(i \in [1,h(T(G))]\). Sections 4.4 and 4.5 show the computation processes when \(i\) is even and odd, respectively. Section 4.6 analyzes the time complexity of the entire algorithm.

### 4.3 Preprocess

In this section, from a given rooted graph \(G = (V,E)\) and its BCL-tree \(T(G)\), we describe how to construct the lists \(\text{root}(i)\) and \(\text{edge}(i)\) for all the odd numbers \(i \in [1,h(T(G))]\).

Executing a bucket sort for the root-vertices \(r_g\), \(B \in \mathcal{B}(i)\) separately for each odd number \(i\) would take \(O(|V| \times |\Lambda^k|) = O(|V|)\) time. This can be reduced to \(O(|V|)\) time. First we initialize the list \(\text{root}(i)\) for each odd number \(i \in [1,h(T(G))]\) by an empty list. Then we execute a bucket sort for all root-vertices \(r_g\) in \(G\) according to their labels \(h(r_g)\). By scanning the sorted list and appending each vertex \(r_g\) at the end of the corresponding lists \(\text{root}(i)\), we construct the lists \(\text{root}(i)\) for all odd numbers \(i \in [1,h(T(G))]\) simultaneously. The bucket sort is executed only once and its time and space complexities are \(O(|V|)\).

We initialize the list \(\text{edge}(i)\) for each odd number \(i \in [1,h(T(G))]\) by an empty list. Then we execute a bucket sort for all edges \(e \in E\) according to their labels \(h(e)\) in \(\Lambda^k\).

By scanning the sorted list and appending each edge at the end of the corresponding list \(\text{edge}(i)\), we construct the lists \(\text{edge}(i)\) for all odd numbers \(i \in [1,h(T(G))]\) at the same time. Again the bucket sort is executed only once and its time and space complexities are \(O(|E|)\).

**Lemma 7:** For a given labeled graph \(G = (V,E)\) with its center, we can construct the sets \(\text{root}(i)\) and \(\text{edge}(i)\) for all odd numbers \(i \in [1,h(T(G))]\) in \(O(|V| + |E|)\) time and space.
4.4 Computation Process when $i$ is Even

In this section, we describe how to compute indices $\text{id}(G_v)$ for all nodes $x \in V(i)$ with an even integer $i \geq 0$, i.e., indices $\text{id}(G_v)$ of all leaf/cut-vertices $v \in V(i)$ from the given list $N_{\text{child}}(i)$ and the indices $\text{id}(G_y)$, $y \in V(j)$ with $j = 0, 1, \ldots, i - 1$.

For $i = 0$, we show how to assign the indices of subgraphs $G_v$ rooted at leaf-vertices $v$ (i.e., vertices corresponding to nodes in $V(0)$). Let $N_{\text{leaf}}(G) \subseteq V$ denote the set of labels of leaf-vertices in $G$. We execute a bucket sort for all leaf-vertices $v$ according to their labels $l(v) \in V$. By scanning the result of sorting, we can get a ranking $\Lambda(v)$ of the subgraph rooted at each leaf-vertex $v$, we set the index $\text{id}(G_v)$ of the subgraph rooted at each leaf-vertex $v$ by $\text{id}(G_v) := \rho_{\text{leaf}}(l(v))$. Obviously, it can be done in $O(|V(G)|)$ time and space.

**Lemma 8:** For an even number $i = 0$, we can compute indices $\text{id}(G_v)$ of the rooted subgraphs $G_v$ for all cut-vertices $v \in V(i)$ in $O(V(i) + |V(i)|)$ time and space.

Let $i \geq 2$. Then nodes in $V(i)$ are C-nodes, and $\text{id}(N_{\text{child}}(i))$ is the set of indices of all the subgraphs $G_B$ trimmed at blocks $B$ corresponding to N-nodes in $N_{\text{child}}(i)$. We can obtain a ranking $\rho_{\text{child}}(i)$ over the set of all $\text{id}(G_B)$ with B-nodes $B \in N_{\text{child}}(i)$ by scanning the sorted list $N_{\text{child}}(i)$. We next encode each subgraph $G_v$ rooted at a vertex $v$ into a string $S(v)$ of integers.

**Definition 9:** Let $B_1, B_2, \ldots, B_d$ be the blocks whose root-vertices are $v$ where $\text{id}(G_{B_1}) \leq \text{id}(G_{B_2}) \leq \ldots \leq \text{id}(G_{B_d})$. Then we define

$$S(v) = (\rho_{\text{child}}(i)(\text{id}(G_{B_1})), \rho_{\text{child}}(i)(\text{id}(G_{B_2})), \ldots, \rho_{\text{child}}(i)(\text{id}(G_{B_d}))).$$

We compute the codes $S(v)$ of subgraphs $G_v$ rooted at vertices $v \in V(i)$, sort them lexicographically. We scan the sorted list of codes and for each $v \in V(i)$ such that $|S(v)| = 1$ holds (i.e., the C-node $v$ has exactly one child in the BCL-tree), and set $\text{id}(G_v) := \text{id}(G_B)$, where $B$ is the block whose root-vertex is $v$. Let $S(i)$ denote the set of codes $S(v)$ for all cut-vertices $v$ corresponding to C-nodes in $V(i) \setminus \overline{N}(i)$. Again we scan the sorted list of codes and assign the ranks of the codes corresponding to C-nodes in $V(i) \setminus \overline{N}(i)$. We set $\text{id}(G_v) := \text{id}_{\text{max}}(i - 1) + \rho_{\text{child}}(i)(S(v))$. The validity of indices that we compute in the above way is guaranteed by the following lemma.

**Lemma 10:** For two cut-vertices $u, v \in V(i)$, it holds that

$$S(u) = S(v) \iff G_u \approx G_v.$$ 

**Proof:** For two B-nodes $B, B' \in N_{\text{child}}(i)$, it holds that $\rho_{\text{child}}(i)(\text{id}(G_B)) = \rho_{\text{child}}(i)(\text{id}(G_{B'}))$ if and only if $G_B \approx G_{B'}$. Then from the definition, the code $S(v)$ of a cut-vertex $v \in V(i)$ is uniquely determined from the structure of $G_v$. Thus rooted-isomorphic graphs always produce the same values of $S$. Conversely, since the structure of $G_v$ can be fully recovered from $S(v)$, non rooted-isomorphic graphs always produce different values of $S$. □

All we have to do is computing the codes $S(v)$ for all cut-vertices $v \in V(i)$ and sorting them lexicographically. Dinitz et al. [3] showed that this can be done in $O(|N_{\text{child}}(i)|)$ time and space using a tree-like data structure which represents each code by a path in the data structure. However, as their data structure and algorithm are rather complex, we introduce another simple way.

Firstly, we compute the codes $S(v)$ for all cut-vertices $v \in V(i)$. After initializing $S(v)$ to be an empty list for each $v \in V(i)$, we can scan the sorted list $N_{\text{child}}(i)$. For each B-node $B \in N_{\text{child}}(i)$, we append $\rho_{\text{child}}(i)(\text{id}(G_B))$ to the end of the list $S(r_B)$ (note that $r_B \in N(i)$). Recall that $N_{\text{child}}(i)$ is the list of B-nodes $B$ which are children of nodes in $N(i)$ sorted in a nondecreasing order by their indices $\text{id}(G_B)$. After scanning all the B-nodes in $N_{\text{child}}(i)$, we get the codes $S(v)$ for all C-nodes $v \in V(i)$. The computation process can be done in $O(|N_{\text{child}}(i)|)$ time and space. After that, we sort all the codes $S(v), v \in V(i)$ lexicographically.

**Lemma 11:** For an even number $i \in [2, h(T(G))]$, assume that the sorted list $N_{\text{child}}(i)$ and the indices $\text{id}(G_v)$ for all $x \in V(j), j = 0, 1, \ldots, i - 1$ are available. Then we can compute indices $\text{id}(G_v)$ of the rooted subgraphs $G_v$ for all cut-vertices $v \in V(i)$ in $O(|N_{\text{child}}(i)|)$ time and space.

**Proof:** At first we compute the ranks of all the nodes $v \in N_{\text{child}}(i)$. Since $N_{\text{child}}(i)$ is given, it can be done in $O(|N_{\text{child}}(i)|)$ time and space. Then we compute $S(v)$ for all C-nodes $v \in V(i)$. This can be done in $O(|N_{\text{child}}(i)|)$ time and space as shown in the above.

Then we sort lexicographically the codes $S(v)$ for all C-nodes in $V(i)$. The sum of the length of codes are $|N_{\text{child}}(i)|$, and each element in the codes is an integer from 1 to at most $|N_{\text{child}}(i)|$ from its definition. It is known that lexicographical sort of such strings can be done in $O(|N_{\text{child}}(i)| + |N_{\text{child}}(i)|) = O(|N_{\text{child}}(i)|)$ time and space, using a modification of radix sort (see Theorem 2).

Thus we can compute indices $\text{id}(G_v)$ for all cut-vertices $v \in V(i)$ in $O(|N_{\text{child}}(i)|)$ time and space. □

4.5 Computation Process when $i$ is Odd

In this section, we describe how to compute indices $\text{id}(G_v)$ for all nodes $x \in V(i)$ with an odd integer $i \geq 0$, i.e., indices $\text{id}(G_v)$ of all the subgraphs trimmed at blocks $B \in B(i)$ from the given lists $\text{root}(i)$, $\text{edge}(i)$, $N_{\text{child}}(i)$, and all the indices $\text{id}(G_v), y \in V(j), j = 0, 1, \ldots, i - 1$.

For $i = h(T(G))$, $N(i)$ consists of exactly one B-node $B$ which corresponds to the center of $G$. Then we set $\text{id}(G_B) := \text{id}_{\text{max}}(i - 1) + 1$.

Let $i < h(T(G))$. We denote by $\Lambda_{\text{root}}(i)$ and $\Lambda_{\text{edge}}(i)$ the sets of labels of vertices in $\text{root}(i)$ and those of edges in $\text{edge}(i)$, respectively. We scan the lists $\text{root}(i)$ and $\text{edge}(i)$ and compute a ranking $\rho_{\text{root}}(i)$ over the set $\Lambda_{\text{root}}(i)$ and the ranking $\rho_{\text{edge}}(i)$ over $\Lambda_{\text{edge}}(i)$, respectively.
Each block \( B \in \mathcal{B}(i) \) is not the center of \( G \) and has its root-vertex \( r_B \). Note that \( Idl(\mathcal{N}_{\text{child}}(i)) \) is the set of indices of subgraphs \( G_i \) rooted at cut-vertices \( v \in V(B) \setminus \{r_B\} \) in all the blocks \( B \in \mathcal{B}(i) \) (i.e., cut-vertices corresponding to C-nodes in \( \mathcal{N}_{\text{child}}(i) \)). We scan the list \( \mathcal{N}_{\text{child}}(i) \) and compute the rank \( \rho_{\text{child}}(\text{id}(G_i)) \) for all \( v \in V(B) \setminus \{r_B\}, B \in \mathcal{B}(i) \).

We denote by \( n_i \) and \( m_i \) the sums of the numbers of vertices and edges in blocks in \( \mathcal{B}(i) \), respectively, i.e., \( n_i := \sum_{B \in \mathcal{B}(i)} |V(B)| \) and \( m_i := \sum_{B \in \mathcal{B}(i)} |E(B)| \). By definition, \( \max(|\mathcal{N}_{\text{root}}(i)|, |\mathcal{N}(\mathcal{N}_{\text{child}}(i))|) \leq n_i \) and \( |\mathcal{A}(i)| \leq m_i \).

We represent each subgraph \( G_B \) trimmed at a block \( B \in \mathcal{B}(i) \) as a block \( B_G \in \mathcal{G}(G, n_i, m_i) \), which is the block \( B \) rooted at its root-vertex \( r_B \) with the following vertex and edge labels.

\[
l(v) := \begin{cases} \rho_{\text{root}}(\text{id}(v)), & \text{if } v \text{ is the root-vertex of } B, \\ \rho_{\text{child}}(\text{id}(G_i)), & \text{otherwise}, \end{cases} \\
\hat{k}(e) := \rho_{\text{child}}(\text{id}(G_i)).
\]

**Lemma 12:** [10] Let \( B \) and \( B' \) be blocks in \( \mathcal{B}(i) \). Then it holds that

\[ B_G \approx B'_G \iff G_B \approx G_{B'} . \]

**Proof Sketch:** For each two cut-vertices \( v \) and \( u \) corresponding to C-nodes in \( \mathcal{N}_{\text{child}}(i) \), it holds that \( \rho_{\text{child}}(\text{id}(G_i)) = \rho_{\text{child}}(\text{id}(G_u)) \) if and only if \( G_v \approx G_u \) holds. Then by definition, for each block \( B \in \mathcal{B}(i) \), the rooted block \( B_G \in \mathcal{G}(G, n_i, m_i) \) is uniquely determined from the structure of \( G_B \).

Thus rooted-isomorphic graphs \( G_B, B \in \mathcal{B}(i) \) always produce the rooted-isomorphic blocks \( B_G \in \mathcal{G}(n_i, m_i) \). Conversely, since the structure of \( G_B \), \( B \in \mathcal{B}(i) \) can be fully recovered from the rooted block, non-rooted-isomorphic graphs \( G_B \) always produce non-rooted-isomorphic blocks \( B_G \in \mathcal{G}(n_i, m_i) \).

Now we use the signature \( \sigma \) for \( (G, n_i, m_i) \) in Definition 3, and we denote \( \sigma(B_G) \) by \( S(B) \) for each block \( B \). From Lemma 2, it holds that

\[ S(B) = S(B') \iff G_B \approx G_{B'} . \]

We compute the signatures \( S(B) \) for all the subgraphs \( G_B \) trimmed at blocks \( B \in \mathcal{B}(i) \), sort them lexicographically, and assign the ranks of them. Let \( S(i) \) denote the set of signatures \( S(B) \) of the rooted subgraphs \( G_B \) for all blocks \( B \in \mathcal{B}(i) \). We assign \( \text{id}(G_B) := \text{id}_{\text{max}}(i-1) + \rho(\sigma(S(B))) \). From the assumption of our class \( \mathcal{G} \) of graphs (Definition 3), we have the following result.

**Lemma 13:** For an odd number \( i \in \{1, h(T(G))\} \), assume that the lists \( \text{root}(i), \text{edge}(i), \mathcal{N}_{\text{child}}(i) \), and the indices \( \text{id}(G_i), x \in \mathcal{N}(i), j = 0, 1, \ldots, i-1 \) are available. Then we can compute indices \( \text{id}(G_B) \) of the subgraphs \( G_B \) for all blocks \( B \in \mathcal{B}(i) \) in \( O(n_i + m_i + M_i + L_i + S_i) \) time and in \( O(n_i + m_i + M_i + L_i + S_i) \) space, where \( M_i = \max_{B \in \mathcal{B}(i)} |\mathcal{N}(B)|, L_i = \sum_{B \in \mathcal{B}(i)} |\mathcal{L}(B)|, T_i = \sum_{B \in \mathcal{B}(i)} |\mathcal{A}(B)|, \) and \( S_i = \max_{B \in \mathcal{B}(i)} |S(B)| \).
(G_o,a,b) be the set of outerplanar blocks B with a vertex labeling and an edge labeling such that each block B is rooted at a vertex r_B ∈ V(B) and the label of each vertex (resp., edge) is chosen from the label set [1,a] (resp., [1,b]). In this section, we introduce a signature S_o of the set (G_o,a,b). The following definition is based on the signature of a rooted outerplanar graph given by Manning and Atallah [14].

The function S_o that maps each rooted outerplanar block B ∈ (G_o,a,b) to a string S_o(B) of integers is defined as follows. Let V(B) = {r_B,v_1,v_2,...,v_m], where π = [r_B,v_1,v_2,...,v_m] denote the order of the vertices that appear along H(B) and ̂π = [r_B,v_m,v_m-1,...,v_1] denote the reverse order of π (where ̂π = π if B is a bridge). For each edge (u,w) in a block B which is not the bridge, let d_e(u,w) (resp., ̂d_e(u,w)) denote the distance from u to w along H(B) in the orientation π (resp., ̂π), and (u,w) denote the label of the edge e = (u,w). In the next definition, we interpret two symbols, (“” and “”) in a string mean integers 0 and max{|V(B)|,a+1}, respectively.

Definition 15: The function S_o that maps a rooted outerplanar block B ∈ (G_o,a,b) to a string S_o(B) is defined as follows.

(i) If B is a bridge composed of a vertex {r_B,w} and an edge e between r_B and w, then we define S_o(B) = [(r_B),l(e),l(w)].

(ii) If B is not a bridge, then we define S_o(B) as follows. Each non-root vertex u ∈ V(B) \ {r_B} is encoded into a string c_u(u) as follows. Let c_u(u) denote the string consisting of the ordered pairs (d_e(u,w),l(u,w)) for all edges (u,w) incident to u, sorted lexicographically. A string S_o(B) of integers is defined by an alternating sequence of the code c_u(u) and the vertex label l(u) after starting with the label l(r_B) of the root r_B, i.e.,

S_o(B) = [(l(r_B),c_π(v_1),l(v_1),c_π(v_2),l(v_2),...,c_π(v_m),l(v_m))].

For the other orientation ̂π, we construct the other codes c_u(u), u ∈ V(B) \ {r_B} and another string S_̂o(B) symmetrically. Let S_o(B) be one of the strings S_o(B) and S_̂o(B) which is lexicographically not smaller than the other.

In the definition, one of two orientations π and ̂π of B is chosen by choosing one string S_o(B) or S_̂o(B).

See Fig. 3 for an example. In Fig. 3, the label of each vertex is drawn as a bold number and the label of each edge is drawn as a multiplicity of the edge. At first, we assign an orientation π of the block B by r_B → v_1 → v_2 → ... → v_4. For the vertex v_1, the edge (v_1,v_2) is encoded as a pair (1,1), the inner edge (v_1,v_3) is encoded as a pair (3,1), and the edge (v_3,v_4) is encoded as a pair (5,1). Then we have c_π(v_1) = ((1,1),(3,1),(5,1)). Similarly, we have c_π(v_2) = ((1,1),(5,1)), c_π(v_3) = ((1,1),(5,1)), c_π(v_4) = ((1,1),(3,1),(5,1)).

Then we have

S_o(B) = [(l(r_B),c_π(v_1),l(v_1),c_π(v_2),l(v_2),c_π(v_3),l(v_3),c_π(v_4),l(v_4),l(v_5))]

= [5,((1,1),(3,1),(5,1)),3,((1,1),(5,1)),1,((1,1),(5,1))], 4,((1,1),(3,1),(5,1)),2,((1,2),(5,1)),1].

Symmetrically, by taking the reverse orientation ̂π, we have

S_̂o(B) = [(5,((1,1),(5,2)),1,((1,1),(3,1),(5,1)),2,((1,1),(5,1))], 4,((1,1),(3,1),(5,1)),1,((1,1),(3,1),(5,1)),3].

As S_o(B) is lexicographically greater than S_̂o(B), we choose the orientation π and define S_o(B) = S_o(B).

We show that the function S_o is a signature which is computable by (|E(B)| = |V(B)|, M(B) = |V(B)|, X(B) = |V(B)|, S(B) = |V(B)|).

Lemma 16: For any blocks B,B′ ∈ (G_o,a,b), it holds that S_o(B) = S_o(B′) if and only if B and B′ are rooted-isomorphic.

Proof: It is known that the Hamilton cycle of an outerplanar block is uniquely determined [15]. Since the code S_o(B) of a rooted labeled outerplanar block B is uniquely determined, rooted-isomorphic graphs always produce the same values of S_o. Conversely, since the structure of G_B can be fully recovered from S_o(B), non-rooted-isomorphic graphs always produce different values of S.

Lemma 17: For any block B ∈ (G_o,a,b), the length |S_o(B)| of the string is O(|V(B)|).

Proof: From the definition, the length of the string S_o(B) is O(|V(B)| + |E(B)|) = O(|V(B)|), as |E(B)| ≤ 2|V(B)| holds for an outerplanar block B [15].

Lemma 18: For any block B ∈ (G_o,a,b), let M be the maximum integer of a + 1, b + 1 and |V(B)|. Then each element of the code S_o(B) is an integer from 0 to at most M.

Proof: From the definition, the components of the string S_o(B) are symbols “,” “)” and integers from 1 to at most max{|V(B)| - 1,a,b}. Thus each component of the signature S_o(B) is an integer from 0 to at most M.
Lemma 19: For any block \( B \in (G_o, a, b) \), we can compute the code \( S_o(B) \) in \( O(|V(B)|) \) time and space.

For proving Lemma 19, we describe an algorithm for computing the code \( S_o(B) \) from a given rooted block \( B \in (G_o, a, b) \). The way of computing the code is based on the way of Manning and Atallah [14]. Note that the vertex encodings \( c_e(v) \) or \( \bar{c}_e(v) \) for all \( v \in V(B) \) are computed simultaneously rather than sequentially. In the following pseudo code, we describe how to compute the code \( S_o(B) \) and \( S_o(B) \) of a rooted block \( B \) in \( O(|V(B)|) \) time and space.

Algorithm \( CalS_o(B) \)

Input: A rooted outerplanar block \( B \in (G_o, a, b) \) and its Hamilton cycle \( H(B) \)

Output: The string \( S_o(B) \)

Assign two orientations \( \pi \) and \( \pi' \) to the Hamilton cycle \( H(B) \);

Let \( V(B) = \{v_1, v_2, \ldots, v_m\} \) and \( v_1, v_2, \ldots, v_m \) are ranged along the orientation \( \pi \), where \( v_1 \) is the root of \( B \);

for each vertex \( v \in V(B) \) do

Initialize \( c_e(v) := \emptyset \);

end for;

for \( i = 1, 2, \ldots, m \) do

for each edge \( e = (u_i, v_i) \) with \( i > j \) do

Append \( (d_e(u_i, v_i)) = i - j, (\hat{h}(u_i, v_j)) \) to \( c_e(v_j) \)

end for;

end for;

for \( i = 1, 2, \ldots, m \) do

for each edge \( e = (u_i, v_i) \) with \( i < j \) do

Append \( (d_e(u_i, v_i)) = |V(B)| - (j - i), (\hat{h}(u_i, v_j)) \) to \( c_e(v_j) \)

end for;

end for;

Set \( S_o(B, G_B) := (\hat{h}(v_1)) \);

for \( i = 2, 3, \ldots, m \) do

Concatenate \( (c_e(v_1), \hat{h}(v_i)) \) to the end of \( S_o(B) \)

end for;

Compute \( S_o(B) \) similarly by considering the orientation \( \bar{\pi} \);

Let \( S_o(B) \) be one of the strings \( S_o(B) \) and \( S_o(B) \) which is lexicographically not smaller than the other.

We show the validity of the above algorithm. For any \( v_i \in V(B) \setminus \{v_1\} \), let \( c_e(v, B) = (a_1, a_2, \ldots, a_q) \) where \( a_k, k = 1, 2, \ldots, q \) is the pair \( (d_e(u_i, v_j), \hat{h}(u_i, v_j)) \) corresponding to the edge \( e \) between \( v_i \) and \( v_j \). From the definition, there exists some integer \( p \in \{0, 1, 2, \ldots, q \} \) such that \( a_k, k = 1, 2, \ldots, p \) is the pair corresponding to the edge between \( v_i \) and \( v_j \) for \( i < j \) and \( a_k, k = p + 1, p + 2, \ldots, q \) is the pair corresponding to the edge between \( v_i \) and \( v_j \) for \( i > j \). In the above algorithm, the first loop appends \( a_1, a_2, \ldots, a_p \) to the initially empty string \( c_e(v) \) and the second loop appends \( a_{p+1}, a_{p+2}, \ldots, a_q \) to the result.

Proof of Lemma 19: For a given outerplanar block \( B \), we can detect the Hamilton cycle of \( B \) in \( O(|V(B)|) \) time and space [15]. Then we assign two orientations \( \pi \) and \( \bar{\pi} \) to the Hamilton cycle and Algorithm \( CalS_o(B) \) computes the code \( S_o(B) \) in \( O(|V(B)|) \) time and space. □

Now we get the following lemma.

Lemma 20: The function \( S_o \) is a signature of the set \( (G_o, a, b) \) which is computable by \( (|V(B)|, |V(B)|, |V(B)|, |V(B)|) \).

Proof: Set \( \mathcal{L}(B) := |V(B)|, \mathcal{M}(B) := |V(B)|, \mathcal{N}(B) := |V(B)| \) and \( S(B) := |V(B)| \). Then from Definition 3, Lemmas 16, 17, 18 and 19, \( S_o \) is a signature which is computable by \( (|V(B)|, |V(B)|, |V(B)|, |V(B)|) \).

Then we have the following result.

Theorem 21: For a given labeled outerplanar graph \( G = (V, E) \) and its center, indices of all the subgraphs rooted at vertices can be computed in \( O(|V|) \) time and space.

Proof: We apply Theorem 14 to outerplanar graphs. As \( G \) is an outerplanar graph, it holds that \( |E| \leq 2|V| - 3 \) [15]. Thus it holds that \( |E| + 1 \leq 2|V| - 1 \). Thus about integers appearing in the computation process, it is enough to assume that integers from 0 to at most \( 2|V| - 1 \) can be stored in unit space and each of addition, subtraction, multiplication, and division over such integers can be executed in unit time. As shown in proof of Lemma 20, set \( \mathcal{L}(B) := |V(B)|, \mathcal{M}(B) := |V(B)|, \mathcal{N}(B) := |V(B)| \) and \( S(B) := |V(B)| \) and now we get the result of this theorem. □

6. Concluding Remarks

In this paper, we have introduced a framework for computing indices of all the subgraphs rooted at vertices in a given rooted graph, which belongs to the particular class of graphs. For the given class of graphs, we assume that a signature of rooted biconnected components with a vertex labeling and an edge labeling is known. In our framework, we guarantee upper and lower bounds of integers used in the computation process. For a given labeled graph \( G = (V, E) \) with its center, let \( M \) be the maximum number of \( 2|V| - 1, |V| + 1 \) and \( |E| + 1 \). In our algorithm, we use only integers from 0 to \( M \) except for the process for computing signatures of blocks. Under the constraints, our algorithm runs in linear time and space except for time and space complexities for computing signatures.

In addition to that, we have introduced a signature of blocks for labeled outerplanar graphs which can be computed in linear time and space. By applying this signature to the framework above, we can compute indices of all the subgraphs rooted at vertices in a given labeled outerplanar graph \( G = (V, E) \) with its center in linear time and space, where we use only integers from 0 to at most \( 2|V| - 1, |V| + 1 \) in the computation process. Our result includes the case when \( G \) is a tree where time complexity and space complexity are similar to the algorithm given by Dinitz et al. [3], though integers appearing in our computation process increases due to consideration of subgraphs trimmed at bridges.

Our algorithm is expected to have applications to other
algorithms which need to repeatedly detect whether two subgraphs are isomorphic or not, especially to algorithms for enumerating stereoisomers of chemical graphs [7]–[9]. It is left as a future work to design a framework for computing indices for subgraphs of a biconnected graph which is composed of triconnected components in a specified class $G$ of triconnected graphs assuming that a procedure for computing a signature for the class $G$ is available.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison-Wesley Longman Publishing, 1974.
[2] H.L. Bodlaender, “Polynomial algorithms for graph isomorphism and chromatic index on partial $k$-trees,” J. Algorithms, vol.11, no.4, pp.631–643, Dec. 1990.
[3] Y. Dinitz, A. Itai, and M. Rodeh, “On an algorithm of Zemlyachenko for subtree isomorphism,” Inf. Process. Lett., vol.70, no.3, pp.141–146, March 1999.
[4] H. Fujitwara, J. Wang, L. Zhao, H. Nagamochi, and T. Akutsu, “Enumerating treelike chemical graphs with given path frequency,” J. Chemical Information and Modeling, vol.48, no.7, pp.1345–1357, June 2008.
[5] J.E. Hopcroft and J.K. Wong, “Linear time algorithm for isomorphism of planar graphs (preliminary report),” Proc. 6th Annual ACM Symposium on Theory of Computing, pp.172–184, 1974.
[6] T. Horvath, T. Akutsu, and S. Wrobel, “A refinement operator for outerplanar graphs,” Proc. 16th International Conference of Inductive Logic Programming, pp.95–97, Aug. 2006.
[7] T. Imada, S. Ota, H. Nagamochi, and T. Akutsu, “Enumerating stereoisomers of tree-structured molecules using dynamic programming,” 29th International Symposium on Algorithms and Computation (ISAAC 2009), LNCS 5878, pp.55–64, Dec. 2009.
[8] T. Imada, S. Ota, H. Nagamochi, and T. Akutsu, “Efficient enumeration of stereoisomers of tree-structured molecules using dynamic programming,” J. Mathematical Chemistry, vol.51, no.11, pp.2788–2807, Nov. 2011.
[9] T. Imada, S. Ota, H. Nagamochi, and T. Akutsu, “Efficient enumeration of stereoisomers of outerplanar chemical graphs using dynamic programming,” Graduate School of Informatics, Kyoto University, Technical Report 2010-018, Dec. 2010.
[10] T. Imada and H. Nagamochi, “Indexing all rooted subgraphs of a rooted graph,” Graduate School of Informatics, Kyoto University, Technical Report 2011-002, Jan. 2011.
[11] Y. Ishida, Y. Kato, L. Zhao, H. Nagamochi, and T. Akutsu, “Branch-and-bound algorithms for enumerating treelike chemical graphs with given path frequency using detachment-cut,” J. Chemical Information and Modeling, vol.50, no.5, pp.934–946, April 2010.
[12] Y. Ishida, L. Zhao, H. Nagamochi, and T. Akutsu. “Improved algorithms for enumerating tree-like chemical graphs,” 19th International Conference on Genome Informatics (GIW 2008), Genome Informatics, vol.21, pp.53–64, Dec. 2008.
[13] E.M. Luks, “Isomorphism of graphs of bounded valence can be tested in polynomial time,” J. Comput. Syst. Sci., vol.25, no.1, pp.42–65, Aug. 1982.
[14] J. Manning and M.J. Atallah, “Fast detection and display of symmetry in outerplanar graphs,” Discrete Appl. Math., vol.39, no.1, pp.13–35, Aug. 1992.
[15] S.L. Mitchell, “Linear algorithms to recognize outerplanar and maximal outerplanar graphs,” Inf. Process. Lett., vol.9, no.5, pp.229–232, Dec. 1979.

Tomoki Imada was born in Hamamatsu, on July 8, 1986. He received the B.E. and M.I. degrees from Kyoto University in 2009 and in 2011, respectively. He is an engineer in Panasonic Corporation.

Hiroshi Nagamochi was born in Tokyo, on January 1, 1960. He received the B.A., M.E. and D.E. degrees from Kyoto University, in 1983, in 1985 and in 1988, respectively. He is a Professor in the Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University. His research interests include network flow problems and graph connectivity problems. Dr. Nagamochi is a member of the Operations Research Society of Japan, the Information Processing Society, and the Japan Society for Industrial and Applied Mathematics.