Constant Time Enumeration of Bounded-Size Subtrees in Trees and Its Application

Kunihiro Wasa[1], Takeaki Uno[2], and Hiroki Arimura[1]

[1] Hokkaido University, Graduate School of Information Science and Technology
N14, W9, Sapporo 060-0814, Japan
{wasa,arim}@ist.hokudai.ac.jp
[2] National Institute of Informatics
2-1-2, Hitotsubashi, Tokyo 101-8430, Japan
uno@nii.jp

Abstract. By the motivation to discover patterns in massive structured data in the form of graphs and trees, we study a special case of the k-subtree enumeration problem, originally introduced by (Ferreira, Grossi, and Rizzi, ESA'11, 275-286, 2011), where an input graph is a tree of n vertices. Based on reverse search technique, we present the first constant amortized time enumeration algorithm that lists all k-subtrees of an input tree in $O(1)$ amortized time per subtree. This result improves on the straightforward application of Ferreira et al's algorithm with $O(k)$ amortized time per subtree when an input is restricted to tree. Finally, we discuss an application of our algorithm to a modification of the graph motif problem for trees.

1 Introduction

By emergence of massive structured data in the form of trees and graphs, there have been increasing demands on efficient methods that discovers many of interesting patterns or regularity hidden in collections of structured data [1, 3, 14, 15]. For instance, the proximity pattern mining problem [10, 11] is a class of such pattern discovery problems, where an algorithm is requested to find all collections of items satisfying proximity constraints in a given discrete structure. For example, the proximity string search problem and the graph motif problem are popular examples of such proximity pattern discovery problems.

In this paper, we consider the *k-subtree enumeration problem*, which is originally introduced by Ferreira, Grossi, and Rizzi [8], where an instance consists of an undirected graph G of n vertices and a positive integer $k \geq 1$, and the task is to find all k-subtrees, a connected and acyclic node subsets consisting of exactly k nodes in G. Ferreira *et al.* [8] presented the first output-sensitive algorithm that lists all k-subtrees in a graph G of size n in $O(sk)$ total time and $O(n)$ space, in other words, in $O(k)$ *amortized time* per subtree, where n is the number of edges of an input graph and s is the number of solutions. However, it has been an open question whether there exists a faster enumeration algorithm that solves this problem.

As a main result of this paper, we present the first *constant amortized time enumeration algorithm* for the k-subtree enumeration problem. Our algorithm lists all k-subtrees of an input tree T of size n in $O(n + s)$ total time and $O(n)$ space, that is, in $O(1)$ amortized time per subtree. Our algorithm is based on reverse search technique, proposed by Avis and Fukuda [4], as in the general algorithm by Ferreira *et al.* [8]. However, unlike the case for general input graphs studied by [8], our algorithm achieves the best possible enumeration complexity in amortized sense, i.e., of constant

amortized time per solution. Finally, we discuss an application of our algorithm to a modification of the graph motif problem for trees.

This achievement is based on careful analysis about the cases for k-subtrees in a tree. We classify k-subtrees into two classes of trees in the specific form, one is the class of so-called *serial trees* and another is that of ordinary *non-serial trees*. Our algorithm has two tree-shaped search routes for classes of serial trees and non-serial trees, separately. The first search route visits each node r of T to find the representative serial tree rooted at r, while the second search route, starting from the serial tree, searches the subspace of all k-subtrees containing r as their root. By interleaving two search route into one tree-shaped route $\mathcal{F}_k(T)$ as a family tree of solutions, the algorithm enumerates all k-subtrees using backtracking on $\mathcal{F}_k(T)$.

To achieve constant amortized time complexity, our algorithm represents the search status for the current subtree S by maintaining two lists, called $\mathsf{DelList}(S)$ and $\mathsf{AddList}(S)$, for the candidates of the nodes to delete and to add. Combining the above techniques together, we obtained a constant amortized time algorithm for the k-subtree problem. This result improves on the straightforward application of the $O(k)$ amortized time algorithm of [8] to trees by giving an $O(1)$ amortized time enumeration algorithm when an input is restricted to tree.

1.1 Related work

There have been increasing interests in pattern discovery problem with proximity constraints. The *k-proximity string matching problem* [11] is the problem of finding all occurrences of a combination P of m-colors within a sliding window of length k on a given input string over a color alphabet. Sadakane and Imai [11] introduced the problem and presented a linear time algorithm for the problem. Kim, Lee, and Park [9] studied an extended version of the k-proximity string matching problem, where each the constraint of the colors is generalized to counting inequality type constraints.

The *graph motif problem* is another example of proximity pattern discovery problems, where an instance is a triple of a vertex colored graph G, a multi-set P of colors, and an integer $k \geq 1$, and the task is to find some connected collection S of nodes of G whose multi-set of colors $C(S)$ is identical to P. Lacroix, Fernandes, and Sagot [10] introduced the problem and presented efficient algorithms with applications to bioinformatics, including a negative result that the graph motif problem is NP-hard in general. Fellows, Fertin, and Vialette [7] showed some hardness result including the NP-hardness for trees of degree 3 when k is unbounded, and some exact algorithms in FPT when k is bounded. Recently, Dondi, Fertin, Vialette [6] studied approximation algorithms for the problem.

Although there are increasing number of studies on the graph motif problem [6, 7, 10], there are few studies that attempt to apply efficient enumeration algorithms to this problem. Ferreira, Grossi, and Rizzi [8] is one of such studies with such a motivation. Recent studies [2, 3, 14, 15] applied efficient enumeration algorithms to discovery and mining of combinatorial structures in the real data sets.

The k-subtree enumeration is closely related to a well-known graph problem of enumerating all spanning trees in an undirected graph G [13]. For this problem, Tarjan and Read [13] first presented an $O(ns)$ time and $O(n)$ space algorithm in 1960's, where n is the number of edges in G. Recently, Shioura, Tamura, Uno [12] presented $O(n+s)$ time and $O(n)$ space algorithm. Unfortunately, it is not easy to extend the algorithms for spanning tree enumeration to subtree enumeration.

1.2 Organization of this paper

The rest of this paper is organized as follows. Section 2 introduces basic definitions and results on trees and enumeration algorithms. Section 3 defines the parent-child relationship among k-subtrees. Section 4 presents our efficient algorithm for enumeration of all k-subtrees of a given tree. Section 5 gives an application of the algorithm. Finally, section 6 concludes this paper.

2 Preliminary

In this section, we give basic definitions and notation for trees and their subtrees. For the definitions not found here, please consult textbooks, e.g., [5]. For a set S, we denote by $|S|$ the number of elements in S. For integers i, j ($i \leq j$), we define the set $[i, j] = \{i, i+1, \ldots, j\}$ of consecutive integers.

2.1 Trees

A *rooted tree* is a directed acyclic graph $T = (V, E, \mathsf{root}) = (V(T), E(T), \mathsf{root}(T))$, where V is a set of *vertexes*, $E \in V^2$ is a set of *directed edges*, and $\mathsf{root}(T) \in V$ is a distinguished node, called the *root* of T. For each directed edge $(u, v) \in E$, we call u the *parent* of v and v a *child* of u. In the rooted tree, we assume that every node v other than the root has the unique parent.

The *size* of T is denoted by $n = |T| = |V|$. A node is a *leaf* if it has no children, and is an *internal node* otherwise. We denote by $\mathsf{Leaves}(T)$ the *set of all leaves* in $|T|$. We say that nodes u and v are *siblings* each other if they have the same parent. For each node v, we denote the unique parent of v by $\mathsf{pa}(v)$, and the set of all children of a node u by $\mathsf{children}(v) = \{\, w \in V \mid (v, w) \in E \,\}$.

For any pair of nodes u and $v \in V$, an *undirected path* between u and v (in the sense of undirected graphs) is a sequence of nodes $\pi = (v_0 = u, v_1, \cdots, v_k = v)$ ($k \geq 0$), where either $(v_{i-1}, v_i) \in E$ or $(v_i, v_{i-1}) \in E$ holds for every $i = 1, \cdots, k$. The path π is a *directed path* if it satisfies that $(v_{i-1}, v_i) \in E$ for every i. We define the ancestor-descendant relation $\preceq$ as follows: If there is a directed path from node u to node v, then we define $u \preceq v$, and say that u is an *ancestor* of v, or v is a *descendant* of u. If $u \preceq v$ but $v \npreceq u$, then we define $u \prec v$ and say that u is a *proper ancestor* of v, or v is a *proper descendant* of u. For any node v, we denote by $T(v)$ the *set of all descendants* of v in T. We say that x and y are *incomparable* w.r.t $\preceq$ if $x \npreceq y$ and $y \npreceq x$.

In this paper, we regard an input tree T of size $n \geq 0$ as an ordered tree as follows. We first assume an arbitrary fixed ordering among siblings. Then, we number all nodes of T from 1 to n by the *DFS-numbering*, which is the preorder numbering in the depth-first search [5] on nodes in T. In what follows, we identify the node and the associated node number, and thus, write $V = \{1, \cdots, n\}$. Thus, we can write $u \leq v$ (or $u < v$, resp.) if the numbering of u is smaller than or equal to (smaller than) that of v. As a basic property of a DFS-numbering, we have the next lemma.

Lemma 1. *For any $u, v \in V$, the DFS-numbering on T satisfies the following properties (i) and (ii):*

(i) if v is a proper descendant of u, i.e., $u \prec v$, then $u < v$ holds.
(ii) If v is a properly younger sibling of u, then $u < v$ holds.
(iii) Suppose that x and y are incomparable w.r.t $\preceq$. For any nodes x', y' such that $x' \succeq x$ and $y' \succeq y$, $x < y$ implies that $x' < y'$.

Let $1 \leq k \leq n = |T|$ be any positive integer. A vertex subset $S \subseteq V$ is said to be *connected* if for every pair of nodes $u, v \in S$, there is some undirected path between u and v. A *k-subset* of T is a vertex subset $S \subseteq V$ that consists of exactly k nodes, i.e., $|S| = k$. A *k-subtree* of T is a connected k-subset $S \subseteq V$. Since T is a tree, the subgraph $T(S)$ induced in S is obviously acyclic as an undirected graph. We identify S and the induced subgraph $T(S)$, a subtree of T. In what follows, we denote by $\mathcal{S}_k = \mathcal{S}_k(T)$ the *family of all k-subtrees* of T. Then, we denote by $\mathsf{root}(S)$ the root of S. Now, we introduce two sets $\mathsf{Leaves}(S)$ and $\mathsf{Border}(S)$ of all *leaves* and all *border nodes*, respectively, by:

$$\mathsf{Leaves}(S) = \{\, x \in S \mid (\forall y \in S)\ y \notin \mathsf{children}(x) \,\}.$$
$$\mathsf{Border}(S) = \{\, y \in \mathsf{children}(x) \mid x \in S, y \notin S \,\} = \mathsf{children}(S) \setminus S.$$

$\mathsf{Leaves}(S)$ and $\mathsf{Border}(S)$ are anti-chain in T w.r.t. $\preceq$. The set $\mathsf{Border}(S)$ is called the *border* of S. $\mathsf{Leaves}(S)$ and $\mathsf{Border}(S)$, respectively, contain the information of the interor and exterior of S at its boundary, and will play important roles in our algorithm.

Any 1-subset is always connected. For $k \geq 2$, we have the next technical lemma on the connectivity of k-subsets. The lemma below is frequently used when we treat modification of subtrees in the following sections.

Lemma 2. *Let $k \geq 2$ and $S \subseteq V$ be any k-subset, i.e., a connected k-subset. Let x, y be nodes such that $\mathsf{root}(S) \prec z$ for $z \in \{x, y\}$.*

(i) If $x \in S$, the set $S \setminus \{x\}$ is connected iff $x \in \mathsf{Leaves}(S)$.
(ii) If $y \notin S$, the set $S \cup \{y\}$ is connected iff $y \in \mathsf{Border}(S)$.
(iii) If $x \in S$ and $y \notin S$, the set $(S \setminus \{x\}) \cup \{y\}$ is connected iff $x \in \mathsf{Leaves}(S)$, $y \in \mathsf{Border}(S)$, and $y \notin \mathsf{children}(x)$.

2.2 Enumeration algorithms

An *enumeration algorithm* is an algorithm that receives an input I and outputs all solutions on I without duplicates [4, 12]. The computational complexity of an enumeration algorithm is measured in the output-sensitive complexity, namely, by the running time per solution. If s is the number of solutions, an enumeration algorithm is said to be of *constant amortized time* if it outputs all solutions in $O(1)$ time per solution, that is, $O(s)$ total time. As a computation model, we adopt the usual RAM [5].

Now, we state our problem below.

Problem 1 (k-subtree enumeration in a tree). Given an input tree T and an integer k, enumerate all the k-subtrees of T.

This problem is a special case of the k-subtree problem, studied by Ferreira, Grossi, and Rizzi [8], where an input graph is a tree. Ferreira *et al.* [8] showed an efficient enumeration algorithm that lists all k-subtrees in $O(k)$ amortized time per subtree for a general class of undirected graphs. Therefore, our goal is to devise an efficient algorithm that lists all k-subtrees in $O(1)$ amortized time per subtree.

3 The parent-child relationship among k-subtrees

Our algorithm is designed based on *reverse search* technique by Avis and Fukuda [4]. In the reverse search technique, we define a tree-shaped search route on solutions, called

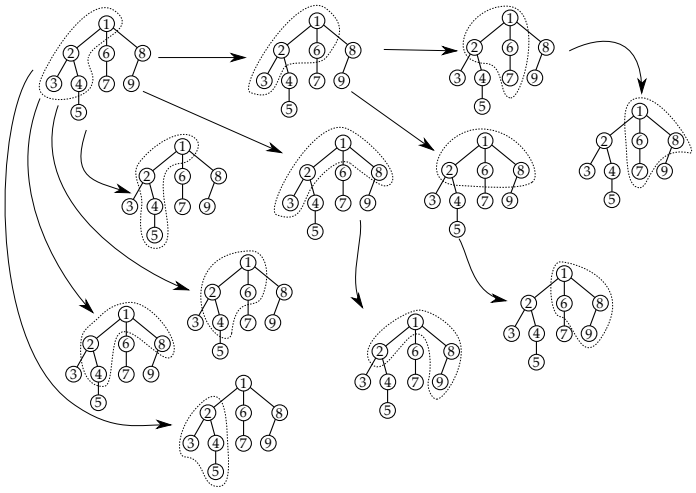

Fig. 1. A family tree for all k-subtrees of an input tree T_1 of size $n = 9$, where $k = 4$.

a *family tree* $\mathcal{F}_k$, as shown in Fig. 1. Then, starting from the root of $\mathcal{F}_k$, we enumerate all solutions using backtracking.

Let $T = (V(T), E(T), \mathsf{root}(T))$ be an input tree of size n, and $1 \leq k \leq n$ be any positive integer. In what follows, we fix T, and we often omit T from expressions if it is clear from contexts. In this section, we give the construction of the family tree $\mathcal{F}_k = \mathcal{F}_k(T)$ for all k-subtrees of T.

First, we introduce the class of serial trees in the following. A node v is said to have at least k descendants if $|T(v)| \geq k$. Let v be any node with at least k descendants. Then, the *serial k-subtree* with root v, denoted by $\mathsf{serial}(v, k)$, is the special subtree of T given by $\mathsf{serial}(v, k) = \{v, v + 1, \ldots, v + k - 1\}$, which consists of k successive nodes starting from v and ends at $v+k-1$. A k-subtree is *serial* if it is a serial k-subtree with some root, and is *non-serial* otherwise. Any serial tree $\mathsf{serial}(v, k)$ satisfies a property that for any nodes $u \leq v \leq w$ of T, if u and w belong to $\mathsf{serial}(v, k)$ then so does v. Clearly, any serial tree can be written as $S = \mathsf{serial}(r, k)$ for $r = \mathsf{root}(S)$.

For example, among the all 4-subtrees of $T = \{1, \ldots, 9\}$ shown in Fig. 1, we have two serial 4-subtrees $\{1, 2, 3, 4\}$ on the top and $\{2, 3, 4, 5\}$ on the bottom. Now, we define the root $\mathcal{I}_k$ of the family tree $\mathcal{F}_k$ for all k-subtrees of T as follows.

Definition 1 (initial tree). The *initial tree* for an input tree T is the serial tree $\mathcal{I}_k = \mathcal{I}_k(T) = \mathsf{serial}(1, k) = \{1, \ldots, k\}$ rooted at the root 1.

Lemma 3. *For any node v with at least k descendants, $S = \mathsf{serial}(v, k)$ is well-defined and a proper k-subtree of T.*

Proof. By definition, all nodes of S are contained in $T(r)$ since $\mathsf{root}(S)$ has at least k descendants. Since the node are numbered in the DFS-numbering, S is obviously a connected subset of V. ∎

Next, we introduce the parent-child relationship P on k-subtrees. In the following, we regard a subset $A \subseteq V(T)$ as the list of nodes which are ordered in the increasing order of their DFS-numbers. Then, we define the head and the tail of the list, respectively, by $\mathsf{head}(A) = \min(A)$ and $\mathsf{tail}(A) = \max(A)$. The following nodes

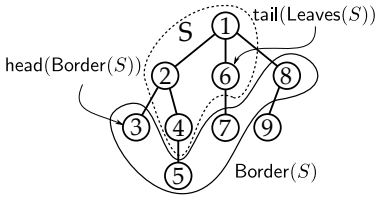

Fig. 2. An example of a 4-subtree $S = \{1, 2, 4, 6\}$ of an input tree $T_1 = \{1, \ldots, 9\}$ of size $n = 9$, where $\mathsf{Border}(S)$, $\mathsf{head}(\mathsf{Border}(S))$, and $\mathsf{tail}(\mathsf{Leaves}(S))$ are the border, the head, and the tail of S, respectively.

of a k-subtree S will be often used in the rest of this paper: $\mathsf{root}(S) = \min(S)$, $\mathsf{tail}(S) = \max(\mathsf{Leaves}(S)) = \max(S)$, and $\mathsf{head}(\mathsf{Border}(S))$.

Clearly, we have the following properties on $\mathsf{root}(S)$, $\mathsf{tail}(S)$, and $\mathsf{head}(\mathsf{Border}(S))$.

Lemma 4 (properties). *For any k-subtree S, we have the following properties:*

(1) For any $x \in \mathsf{Leaves}(S)$ and any $y \in \mathsf{Border}(S)$, $x \in S$ $y \notin S$, and thus $x \neq y$.
(2) $\mathsf{Leaves}(R) \subseteq T(\mathsf{root}(R))$ and $\mathsf{Border}(R) \subseteq T(\mathsf{root}(R))$.

Proof. Since $x \in S$ $y \notin S$, we have Property 1. Property 2 is easy. ■

Lemma 5 (DFS-numbering lemma). *For any k-subtree S, if $h = \mathsf{head}(\mathsf{Border}(S))$, $t = \mathsf{tail}(\mathsf{Leaves}(S))$, and $r = \mathsf{root}(S)$, then*

(a) If S is non-serial, then $h < t$.
(b) If S is serial, then $t < h$. More precisely, $h = t + 1 = r + k$.

Proof. For the proof, please consult the appendix. ■

Example 1. In Fig. 3, we show an examples of a 4-subtree $S = \{1, 2, 4, 6\}$ with $k = 4$ of an input tree T_1 with size $n = 9$. In the figure, the border of S is $\mathsf{Border}(S) = \{3, 5, 7, 8\}$, and the head and tail are given by $\mathsf{head}(\mathsf{Border}(S)) = \min(\mathsf{Border}(S)) = \min\{3, 5, 7, 8\} = 3$ and $\mathsf{tail}(\mathsf{Leaves}(S)) = \max(S) = \max\{1, 2, 4, 6\} = 6$, respectively. Since S is not a serial tree, we obtain the parent 4-subtree $\mathsf{P}(S) = (S \setminus \{6\}) \cup \{3\} = \{1, 2, 3, 4\}$ from S by deleting $\mathsf{tail}(\mathsf{Leaves}(S)) = 6$ and adding $\mathsf{head}(\mathsf{Border}(S)) = 3$.

Now, we define the parent function $\mathsf{P} : \mathcal{S}_k(T) \setminus \{\mathcal{I}_k\} \to \mathcal{S}_k(T)$ for the family $\mathcal{S}_k(T)$ of all k-subtrees of T as follows.

Definition 2 (parent k-subtree). For any k-subtree S of T such that $S \neq \mathcal{I}_k$, the *parent* of S (or the *parent*, for short) is defined by the subset

$$\mathsf{P}(S) = (S \setminus \{y\}) \cup \{x\}$$

obtained from S by deleting a node $y \in S$ and adding a node $x \notin S$, where the nodes y and x are defined in the following cases:

- (*Type I*): If S is non-serial we define $y = \mathsf{tail}(\mathsf{Leaves}(S))$ and $x = \mathsf{head}(\mathsf{Border}(S))$.
- (*Type II*): If S is serial, that is, $S = \mathsf{serial}(\mathsf{root}(S), k)$, then we define $y = \mathsf{tail}(S) = \mathsf{tail}(\mathsf{Leaves}(S))$ and $x = \mathsf{pa}(\mathsf{root}(S))$.

For every types $\tau \in \{I, II\}$, if $\mathsf{P}(S)$ is a parent of S of type τ, then we say that S is a child of $\mathsf{P}(S)$ of type τ.

Lemma 6. *For any k-subtree S of T such that $S \neq \mathcal{I}_k$, $\mathsf{P}(S)$ is connected, and thus, $\mathsf{P}(S)$ is a k-subtree of T.*

Proof. For the proof, please consult the appendix. ■

Let $\mathcal{S}_k = \mathcal{S}_k(T)$ be the class of all k-subtrees of T. The *family tree* for is a directed graph $\mathcal{F}_k = \mathcal{F}_k(T) = (\mathcal{S}_k(T), \mathsf{P}(\cdot), \mathcal{I}_k(T))$, where $\mathcal{V} = \mathcal{S}_k$ is a set of vertexes, $\mathsf{P}(\cdot)$ defines a set of the reverse edges from child k-subtrees to their parents, and $\mathcal{I}_k = \mathcal{I}_k(T) \in \mathcal{V}$ is the root of $\mathcal{F}_k$.

Lemma 7. *For any k-subtree S of T such that $S \neq \mathcal{I}_k$, $\mathcal{F}_k(T)$ forms a spanning tree over $\mathcal{S}_k(T)$.*

Proof. By definition of $\mathsf{P}(\cdot)$, every node except $\mathcal{I}_k$ has exactly one parent. Thus, it is sufficient to show that $\mathcal{F}_k(T)$ is connected and acyclic. We define the weight $f(S)$ of a k-subtree S by the sum $f(S) = \sum_{v \in S} v$ of the DFS-number of its nodes. Firstly, $f(\mathcal{I}_k) = \sum_{i=1}^{k} i = k(k+1)/2 = O(k^2)$ since $\mathcal{I}_k = \{1, 2, \ldots, k\}$. Next, we show that the values of f is strictly decreasing during applications of $\mathsf{P}(\cdot)$. Let $\mathsf{P}(S)$ be the parent of S with the associated nodes x and y. If S is non-serial, we have $y = \mathsf{tail}(\mathsf{Leaves}(S))$ and $x = \mathsf{head}(\mathsf{Border}(S))$. From Lemma 5, we have $x < y$. If S is serial, we have $y = \mathsf{tail}(\mathsf{Leaves}(S))$ and $x = \mathsf{pa}(\mathsf{root}(S))$. From (i) of Lemma 1, we have $x < y$. Overall, since $f(\mathsf{P}(S)) = f(S) - y + x$, we now have $f(\mathsf{P}(S)) < f(S)$. Since $f(\mathcal{I}_k) = O(k^2)$, starting from S, we reach $\mathcal{I}_k$ after at most $O(k^2)$ applications of $\mathsf{P}(\cdot)$. Therefore, the claim is proved. ■

Then, starting from the root $\mathcal{I}_k$, we can enumerate all solutions by depth-first search on $\mathcal{F}_k(T)$ using backtracking.

Example 2. In Fig. 1, we show an example of the family tree $\mathcal{F}_4$ for the class of all 4-subtrees of an input tree T_1 of size $n = 9$. In the figure, the tree S at the upper left corner is the initial tree $\mathcal{I}_k$, and the parent function represents the reverse edges. Then, we can observe that all k-subtrees are reachable from $\mathcal{I}_k$ by following the edges of $\mathcal{F}_4$.

4 The constant amortized time enumeration algorithm

In this section, we present an efficient backtracking algorithm that enumerates all k-subtrees of an input tree T in constant amortized time.

We have introduced in the previous section the family tree $\mathcal{F}_k(T)$ for the class $\mathcal{S}_k(T)$ constructed by the parent pointer $\mathcal{P}$. Then, the remaining task is to develop an efficient method for computing the children of each k-subtree by inverting $\mathcal{P}$.

To compute a child S from a given parent R, where S and R are k-subtrees, it is sufficient to perform the inverse process of computing the parent $R = \mathsf{P}(S)$ from the child S by $R = (S \setminus \{y\}) \cup \{x\}$. This is done by deleting an existing node $x \in R$, called a *delete node*, from R and by adding a new node $y \notin R$, called a *add node*, to R for appropriate candidates for x and y. That is, a child S is generated by the formula

$$S = (R \setminus \{x\}) \cup \{y\}$$

with appropriate constraints on x and y. Below, we consider the details of generation of children according to the types of the generated child S, namely, the case for a *serial subtree* (*Type I*), and the case for a *non-serial subtree* (*Type II*).

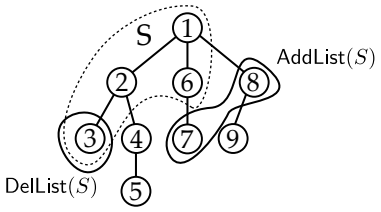

Fig. 3. Examples of $\mathsf{DelList}(S)$ and $\mathsf{AddList}(S)$ for a 4-subtrees, where $\mathsf{DelList}(S) = \{3\}$ and $\mathsf{AddList}(S) = \{7, 8\}$.

4.1 Generation of non-serial k-subtrees

We first consider the case that the generated child k-subtree S is a non-serial subtree (*Type I*).

Definition 3. we define the candidate sets $\mathsf{DelList}(R)$ and $\mathsf{AddList}(R)$ for deleting nodes x and adding nodes y, respectively, as follows.

$$\mathsf{DelList}(R) = \{x \in \mathsf{Leaves}(R) \mid x < \mathsf{head}(\mathsf{Border}(R))\},$$
$$\mathsf{AddList}(R) = \{y \in \mathsf{Border}(R) \mid y > \mathsf{tail}(\mathsf{Leaves}(R))\},$$

where $\mathsf{DelList}(R)$ is the set of all leaves of R that are strictly smaller than $\mathsf{head}(\mathsf{Border}(R))$, and $\mathsf{AddList}(R)$ is the set of all border nodes of R that are strictly larger than $\mathsf{tail}(\mathsf{Leaves}(R))$.

Definition 4. Given a k-subtree R, we define the k-subtree

$$\mathsf{Child}_1(R, x, y) = (R \setminus \{x\}) \cup \{y\},$$

where x and y are any nodes satisfying $x \in \mathsf{DelList}(R)$, $y \in \mathsf{AddList}(R)$, and, $y \notin \mathsf{children}(x)$.

In this case, we say that R is a *parent* of S with type I, or S is a *child* of R with type I. From, Lemma 5, we can prove the following lemma.

Lemma 8. *Let R be a k-subtree of T. For any $x \in \mathsf{DelList}(R)$ and $y \in \mathsf{AddList}(R)$, if $y \notin \mathsf{children}(x)$, then $S = \mathsf{Child}_1(R, x, y)$ is also a proper k-subtree of T. Furthermore, S is a non-serial k-subtree.*

Proof. We have $x \in \mathsf{Leaves}(R)$. If $y \notin \mathsf{children}(x)$, then we have $y \in \mathsf{Border}(R \setminus x)$. Therefore, S is connected from Lemma 2. ■

Theorem 1 (correctness of Child_1). *Let R and S be any k-subtree of T. Suppose that S is non-serial. Then, the following conditions (1) and (2) are equivalent:*

(1) $\mathsf{P}(S) = R$.
(2) $S = \mathsf{Child}_1(R, x, y)$ *for some* $x \in \mathsf{DelList}(R)$ *and* $y \in \mathsf{AddList}(R)$, *and* $y \notin \mathsf{children}(x)$.

Proof. For the proof, please consult the appendix. ■

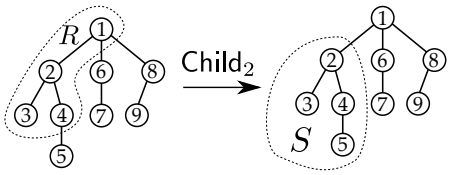

Fig. 4. An example of a serial 4-subtree $S = \mathsf{Child}_2(R)$ generated from a given pre-serial 4-subtree R in T.

4.2 Generation of serial k-subtrees

Next, we consider the special case to generate a serial subtree as a child k-subtree S of a given parent k-subtree (*Type II*). We introduce the class of pre-serial k-subtrees.

Definition 5 (pre-serial tree). A k-subtree R is a *pre-serial subtree* if $\mathsf{root}(R)$ has only one child v such that $|T(v)| \geq k$, and v satisfies that $R(v)$ is a serial $(k-1)$-subtree of T with root v.

In other words, a pre-serial k-subtree R is the k-subtree obtained from some serial $(k-1)$-subtree R' by attaching the new root as the parent of $v = \mathsf{root}(R')$ (Fig.4). Then, we observe that R is a pre-serial k-subtree of T if and only if $\mathsf{root}(R)$ has a single child v and the equality $\mathsf{tail}(\mathsf{Leaves}(R)) = \mathsf{root}(R) + k - 1 = v + k - 2$ holds. Thus, we have the next lemma.

Lemma 9. *We can decide in $O(1)$ time if a given k-subtree S is serial.*

Definition 6. *For any pre-serial k-subtree R with $\mathsf{AddList}(S) \neq \emptyset$, we define*

$$S = \mathsf{Child}_2(R) = (R \setminus \{x\}) \cup \{y\},$$

where $x = \mathsf{root}(R)$ and $y = \min(\mathsf{AddList}(R))$.

In the above definition, the choice of x and y are unique for given R. Then, we can show the next lemma since $y = \mathsf{tail}(R) + 1$ holds when R is pre-serial.

Lemma 10. *For any pre-serial k-subtree R, $S = \mathsf{Child}_2(R)$ is a serial k-subtree of T.*

Theorem 2 (correctness of Child_2). *Let R and S be any k-subtrees of T. Then, the following (1) and (2) hold.*

(1) If R is pre-serial, then (i) $S = \mathsf{Child}_2(R)$ implies (ii) $R = \mathsf{P}(S)$.
(2) If S is serial, then (ii) $R = \mathsf{P}(S)$ implies (i) $S = \mathsf{Child}_2(R)$.

Proof. For the proof, please consult the appendix. ■

4.3 The proposed algorithm

In Algorithm 1, we present the main procedure EnumSubTreeConstantMain and a subprocedure EnumSubTreeConstant the proposed algorithm that enumerates all k-subtrees in a given input rooted tree T of size n in constant amortized time per subtree based on the above discussion.

Algorithm 1 Enumerate all k-subtrees of a rooted tree T

```
1: procedure EnumSubTreeConstantMain(T, k)
2:     Input: T: a rooted tree of size n, k: size of subtrees (1 ≤ k ≤ n);
3:     Number the nodes of T by the DFS-numbering;
4:     Compute the initial k-subtree I_k;
5:     Compute the lists DelList(I_k) and AddList(I_k);
6:     EnumSubTreeConstant(I_k, DelList(I_k), AddList(I_k), T, k);

7: procedure EnumSubTreeConstant(R, DelList(R), AddList(R), T, k)
8:     Input: R: a k-subtree, T: an input tree, k: size;
9:     Print R;
10:    //Case for generating non-serial trees (Sec. 4.1)
11:    for each x ∈ DelList(R) do
12:        for each y ∈ AddList(R) such that y ∉ children(tail(Leaves(R))) do
13:            S ← Child_1(R, x, y);
14:            (DelList(S), AddList(S)) ← Update1(DelList(R), AddList(R), x, y);
15:            EnumSubTreeConstant(S, DelList(S), AddList(S), T, k);
16:    if R is a k-pre-serial tree then
17:        //Case for generating serial trees (Sec. 4.2)
18:        S ← Child_2(R)
19:        (DelList(S), AddList(S)) ← Update2(DelList(R), AddList(R));
20:        EnumSubTreeConstant(S, DelList(S), AddList(S), T, k);
```

Given an input tree T, starting from the initial subtree $\mathcal{I}_k$, the subprocedure EnumSubTreeConstant recursively computes all child k-subtrees S from a given parent k-subtree R by the subtree generation algorithm described in Sec. 4. For any current k-subtree R, the algorithm computes the corresponding non-serial k-subtree as a child of type I. In addition, if R is a pre-serial subtree, then the algorithm computes the corresponding serial k-subtree as a child of type II. Iterating this process, the algorithm EnumSubTreeConstantMain enumerates all k-subtrees in T.

Another key of constant amortized time enumeration is incremental computation of the candidate lists by scanning, which is done as follows. In Algorithm 2 and Algorithm 3, we show the subprocedure Update1 and Update2 that work as follows. Firstly, we consider the subprocedure Update1, which updates the lists $\mathsf{DelList}(S)$ and $\mathsf{AddList}(S)$ by the following recurrence formulas:

$$\mathsf{DelList}(S) = (\mathsf{DelList}(R) \setminus \Delta^-) \cup \Gamma^-,$$
$$\mathsf{AddList}(S) = (\mathsf{AddList}(R) \setminus \Delta^+) \cup \Gamma^+,$$

where the *difference sets* are given below: $\Delta^- = \{\, z \in \mathsf{DelList}(R) \mid x \leq z \,\}$. $\Gamma^- = \{\mathsf{pa}(x)\}$ if $\mathsf{pa}(x) \in \mathsf{Leaves}(S)$, and $\Gamma^- = \emptyset$ otherwise. $\Delta^+ = \{\, z \in \mathsf{AddList}(R) \mid z \leq y \,\}$. $\Gamma^+ = \mathsf{children}(y)$.

From Lemma 1, we can show that $\mathsf{pa}(x)$ is the largest in $\mathsf{DelList}(S)$, and that any node in $\Gamma^+ = \mathsf{children}(y)$ precedes any nodes in $\mathsf{AddList}(R) \setminus \Delta^+$ by definition of DFS-numbering.

Secondly, we consider the subprocedure Update2, which updates the lists $\mathsf{DelList}(S)$ and $\mathsf{AddList}(S)$ by the following recurrence formulas:

$$\mathsf{DelList}(S) = \mathsf{Leaves}(S),$$
$$\mathsf{AddList}(S) = (\mathsf{AddList}(R) \setminus \Delta^+) \cup \Gamma^+,$$

Algorithm 2 Update $\mathsf{DelList}(S)$ and $\mathsf{AddList}(S)$ for generation of Type I

```
1: proc UPDATE1(DelList, AddList, x, y)
   //note: x ∈ DelList, y ∈ AddList;
   //note: x = head(Border(S)),
   //note: y = tail(Leaves(S)).
2: DelList ← DelList \ [x, ∞];
   //Delete x and all the following
   //elements in DelList.
3: if pa(x) is a leaf in S then
4:     DelList ← DelList ∘ (pa(x));
5: AddList ← AddList \ [−∞, y];
6: if y is an internal node in T then
7:     AddList ← children(y) ∘ AddList;
8: return (DelList, AddList);
```

Algorithm 3 Update $\mathsf{DelList}(S)$ and $\mathsf{AddList}(S)$ for generation of Type II

```
1: proc UPDATE2(DelList, AddList, x, y)
   //note: x = root(R).
   //note: y = head(Border(R)).
   //v is the unique child of
   //x = root(R)
2: v ← leftmostchild_R(x);
3: q ← rightsibling_R(v);
4: DelList ← Leaves(S);
5: AddList ← AddList \ [q, ∞];
   //Delete q and all the following q.
6: if y is an internal node in T then
7:     AddList ← children(y) ∘ AddList;
8: return (DelList, AddList);
```

where $\Delta^+ = \{\, z \in \mathsf{AddList}(R) \mid q \leq z \,\}$, where q is the immediate younger sibling (the successor) of v is the unique child of $\mathsf{root}(R)$ (*1). $\Gamma^+ = \mathsf{children}(y)$. (*2).

In the above cases, we can implement the update operations by assuming the left-most child (LC) and right sibling (RS) representation of a rooted tree, where each node has a copy of LC and RS pointers for a subset R in addition to the standard pointers of LC and RS for an input tree T, and they are dynamically redirected when a node is deleted or added to R.

When a recursive procedure call is made, we apply a constant number of operations on candidate lists and record them on a stack, and when the procedure comes back, we apply the inverse of the recorded operations on the lists to reclaim the running state in constant time. For operations scanning more than one nodes, we can amortize the cost by charging it to each updated member of the list because at least one solution can be associated to each member.

From the discussions so far, we have the main theorem on the correctness and the time complexity as follows.

Theorem 3 (constant amortized time enumeration of k-subtree problem). *Given an input rooted tree T of size n, and a positive integer $k \geq 1$, Algorithm1 solves the k-subtree enumeration problem that lists all k-subtrees in $O(1)$ amortized time per subtree using $O(n)$ preprocessing and space.*

This result improves on the straightforward application of Ferreira et al's algorithm [8] with $O(k)$ amortized time per subtree when an input is restricted to tree.

5 Application to the graph motif problem for trees

Let $C = \{1, \ldots, m\}$ $m \geq 1$ be an alphabet of *colors*. A *multi-set X of colors* is represented a function $f_X : C \to \mathbf{N}$, called the *color histogram*, where for every $c \in C$, $f_X(c)$ denotes the number of occurrences, or multiplicity, of c in X. Then, the *graph motif problem* is the problem of, given a triple of a vertex colored graph $G = (V, E)$, a multi-set P of colors with $||P|| = k$, and an integer $k \geq 1$, to find a connected k-subset $S \subseteq V$ of nodes whose multi-set of colors $C(S)$ is identical to P, that is, $f_{C(S)}(c) = f_P(c)$ for every $c \in C$.

From Theorem 3, we have the following result on the restricted version of the graph motif problem for the class of trees.

Theorem 4. *Given an input tree T of size n, a multi-set P of colors with size m, and a positive integer $k \geq 1$, the graph motif problem for tree is solvable in $O(s+n+m)$ total time using $O(n)$ space.*

Proof. For the proof, please consult the appendix. ■

We can easily show the matching upper and lower bounds of the number s of all k-trees in a tree of size n is $s = n^{\Theta(k)}$, where the upper bound is given by counting and the lower bound is shown on a tree consisting only of a root and $(n-1)$ leaves. Although the $O(mn^k)$ total time complexity of a straightforward exhaustive search algorithm on $O(n^k)$ candidate k-subtrees is asymptotically same to that of our algorithm in n, our algorithm can be faster when the number s is much smaller than $n^{\Theta(k)}$ and k is relatively large.

6 Conclusion

In this paper, we studied the k-subtree enumeration problem in rooted trees. As a main result, we presented an efficient algorithm. That solve this problem in constant amortized time per subtree. We also discussed application to graph motif problem.

References

1. S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann, 1999.
2. H. Arimura and T. Uno. Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In *Proc. SDM'09*, pages 1087–1098. SIAM, 2009.
3. T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa. Efficient substructure discovery from large semi-structured data. In *SDM*, 2002.
4. D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1993.
5. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* The MIT Press, 2 edition, 2001.
6. R. Dondi, G. Fertin, and S. Vialette. Finding approximate and constrained motifs in graphs. In *Proc. CPM'11*, volume 6661 of *LNCS*, pages 388–401, 2011.
7. M. Fellows, G. Fertin, D. Hermelin, and S. Vialette. Sharp tractability borderlines for finding connected motifs in vertex-colored graphs. In *Proc. ICALP'07*, volume 4596 of *LNCS*, pages 340–351, 2007.
8. R. Ferreira, R. Grossi, and R. Rizzi. Output-sensitive listing of bounded-size trees in undirected graphs. In *Proc. ESA'11*, volume 6942 of *LNCS*, pages 275–286, 2011.
9. S.-R. Kim, I. Lee, and K. Park. A fast algorithm for the generalized k-keyword proximity problem given keyword offsets. *Inf. Process. Lett.*, 91(3):115–120, 2004.
10. V. Lacroix, C. G. Fernandes, and M.-F. Sagot. Motif search in graphs: Application to metabolic networks. *IEEE/ACM Trans. Comput. Biol. Bioinformatics*, 3:360–368, 2006.
11. K. Sadakane and H. Imai. Fast algorithms for k-word proximity search. *IEICE Trans. Fundam. Electron., Comm., and Comp.*, E84-A(9):2311–2318, 2001.
12. A. Shioura, A. Tamura, and T. Uno. An optimal algorithm for scanning all spanning trees of undirected graphs. *SIAM J. Comput.*, 26(3):678–692, 1997.
13. R. E. Tarjan and R. C. Read. Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. *Networks*, 5(3):237–252, 1975.
14. T. Uno, T. Asai, Y. Uchida, and H. Arimura. An efficient algorithm for enumerating closed patterns in transaction databases. In *Discovery Science*, pages 16–31, 2004.
15. M. J. Zaki. Efficiently mining frequent trees in a forest. In *KDD*, pages 71–80, 2002.

A Appendix

Section 2

Lemma 2. Let $k \geq 2$ and $S \subseteq V(T)$ be any k-subset, i.e., a connected k-subset. Let x, y be nodes such that $\mathsf{root}(S) \prec z$ for $z \in \{x, y\}$.

(i) If $x \in S$, the set $S \setminus \{x\}$ is connected iff $x \in \mathsf{Leaves}(S)$.
(ii) If $y \notin S$, the set $S \cup \{y\}$ is connected iff $y \in \mathsf{Border}(S)$.
(iii) If $x \in S$ and $y \notin S$, the set $(S \setminus \{x\}) \cup \{y\}$ is connected iff $x \in \mathsf{Leaves}(S)$, $y \in \mathsf{Border}(S)$, and $y \notin \mathsf{children}(x)$.

Proof. Claims (i) and (ii) are obvious. Next, we show the if-part of (iii). By (i), if $x \in \mathsf{Leaves}(S)$ then $S' = S \setminus \{x\}$ is connected. By assumption $y \notin \mathsf{children}(x)$, we can show that $y \in \mathsf{Border}(S')$. By (ii), we see that $S' \cup \{y\}$ is connected. The only-if-part of (iii) is easily proved. ■

Let $I(S) = [\mathsf{root}(S), \mathsf{tail}(S)] \subseteq [1, n] = \{1, \ldots, n\}$ be the interval of DFS-numbers that S occupies. The next lemma is useful to prove (a) of Lemma 5 below.

Lemma 11. *Let S be any subtree of T. If $z \notin S$ for some $z \in I(S)$, then there exists some border node $x \in \mathsf{Border}(S)$ such that $x \in I(S)$.*

Proof. If $z \in I(S)$ for some $z \notin S$, then there is some leaf $y \in \mathsf{Leaves}(S)$ such that $y \preceq z$. By climbing up the unique path π from z to y, we can reach a border node $x \in \mathsf{Border}(S)$ on π that is a child of y. Therefore, the lemma is proved. ■

Lemma 5. (DFS-numbering lemma). For any k-subtree S, if $h = \mathsf{head}(\mathsf{Border}(S))$, $t = \mathsf{tail}(\mathsf{Leaves}(S))$, and $r = \mathsf{root}(S)$, then

(a) If S is non-serial, then $h < t$.
(b) If S is serial, then $t < h$. More precisely, $h = t + 1 = r + k$.

Proof. (a): Suppose that S is non-serial. By definition, we have some $z \notin S$ such that $z \in I(S)$. From Lemma 11, there is some $x \in \mathsf{Border}(S)$ such that $x \in I(S)$, too. This implies that $h \leq t$. Since $h \neq t$, we have $h < t$.

(b): From the definitions of $\mathsf{Border}(S)$ and $r = \mathsf{root}(S)$, $r \preceq h$. Thus, it follows from Lemma 1 that $r \leq h$ holds. Since S is serial, $S = [r, t]$ holds, where $t = r + k - 1$. Clearly, $h \notin S$. Therefore, we have $t < h$. Moreover, we can show that $t + 1 \notin S$ and $t+1$ is a child of some node in S, i.e., $t+1$ is a border node of S. Since h is the smallest of such border nodes, we conclude that $h = t + 1$. ■

From Lemma 5 above, we see that$S$ is serial iff $\mathsf{tail}(\mathsf{Leaves}(S)) < \mathsf{head}(\mathsf{Border}(S))$, where $\mathsf{tail}(\mathsf{Leaves}(S)) \neq \mathsf{head}(\mathsf{Border}(S))$ obviously holds.

Section 3

Lemma 6. For any k-subtree S of T such that $S \neq \mathcal{I}_k$, $\mathsf{P}(S)$ is connected, and thus, $\mathsf{P}(S)$ is a k-subtree of T.

Proof. Let S be any k-subtree. Let $R = \mathsf{P}(S) = (S \setminus \{y\}) \cup \{x\}$ be the parent of S, where x and y will be defined below according to the type of $\mathsf{P}(S)$. There are two cases below.

(Case of Type I): If S is non-serial, then $\hat{x} = \mathsf{head}(\mathsf{Border}(S))$ and $\hat{y} = \mathsf{tail}(\mathsf{Leaves}(S))$. From Lemma 5, since S is non-serial, we have $\hat{x} < \hat{y}$. Suppose by contradiction that $\hat{x} \in \mathsf{children}(\hat{y})$. Then, we have $\hat{y} < \hat{x}$ from Lemma 1. From the contradicts, Thus, we have $\hat{x} \notin \mathsf{children}(\hat{y})$. Next, we put $x = \hat{y}$ and $y = \hat{x}$ to apply Lemma 2. Obviously, x is a leaf and y is a border node. From (iii) of Lemma 2, we know that $\mathsf{P}(S)$ is connected. (End of Proof for Type I)

(Case of Type II): If S is serial, then $\hat{y} = \mathsf{tail}(\mathsf{Leaves}(S))$ and $\hat{x} = \mathsf{pa}(\mathsf{root}(S))$. Since S is not $\mathcal{I}_k$, $\hat{x}$ is defined. We put $x = \hat{y}$ and $y = \hat{x}$. Since $x = \hat{y}$ is a leaf of S, (i) of Lemma 2 shows that $S' = (S \setminus \{x\})$ is connected. Since $y = \hat{x}$ is adjacent to $\mathsf{root}(S)$, clearly, $\mathsf{P}(S) = (S' \cup \{y\})$ is connected. (End of Proof for Type II) ■

Section 4

The following technical lemma is easy, but useful and frequently used in showing the correctness of our algorithm. For simplicity, we abbreviate $h(X) = \mathsf{head}(X)$, $t(X) = \mathsf{tail}(X)$, $B(S) = \mathsf{Border}(S)$, and $L(S) = \mathsf{Leaves}(S)$ in the followings.

Lemma 12. *Let R, S be two sets with $|R| = |S| = k$. Let*

$$S = (R \setminus \{x\}) \cup \{y\}, \quad (x \in R, y \notin R), \tag{1}$$
$$R' = (S \setminus \{\hat{y}\}) \cup \{\hat{x}\}, \quad (\hat{x} \notin S, \hat{y} \in S). \tag{2}$$

Then, (1) $R = R'$ iff $x = \hat{x}$ and $y = \hat{y}$.

By the definitions $\mathsf{DelList}(R)$ and $\mathsf{AddList}(R)$, we have $x \in \mathsf{DelList}(R)$ iff $x \in \mathsf{Leaves}(R)$ and $x < \mathsf{head}(\mathsf{Border}(R))$. Similarly, we have $y \in \mathsf{AddList}(R)$ iff $y \in \mathsf{Border}(R)$ and $y > \mathsf{tail}(\mathsf{Leaves}(R))$. Now, we show the correctness of $\mathsf{Child}_1(\cdot)$ and $\mathsf{Child}_2(\cdot)$ as follows.

Lemma 13 (completeness of Child_1). *Let R and S be any k-subtree of T. Suppose that S is non-serial. (1) $\mathsf{P}(S) = R$ holds only if (2) $S = \mathsf{Child}_1(R, x, y)$ for some $x \in \mathsf{DelList}(R)$ and $y \in \mathsf{AddList}(R)$ such that $y \notin \mathsf{children}(x)$.*

Proof. Suppose that $R = \mathsf{P}(S)$. By definition, we have $R = (S \setminus \{\hat{y}\}) \cup \{\hat{x}\}$ for the nodes $\hat{x} = h(B(S))$ and $\hat{y} = t(L(S))$. Suppose we set $x = \hat{x}$ and $y = \hat{y}$. Then, the remaining task is to show that these x and y satisfy the pre-condition for the operation $\mathsf{Child}_1(R, x, y)$ as in the following. First, we show the membership of $\hat{x}$ and $\hat{y}$ to $\mathsf{DelList}(R)$ and $\mathsf{AddList}(R)$ as follows. By the construction of $R = \mathsf{P}(S)$, we have the recurrences

$$L(R) = (L(S) \setminus \{\mathsf{pa}(\hat{x}), \hat{y}\}) \cup \{\hat{x}, \mathsf{pa}(\hat{y})\}, \tag{3}$$
$$B(R) = (B(S) \setminus (\{\hat{x}\} \cup \mathsf{children}(\hat{y}))) \cup (\mathsf{children}(\hat{x}) \cup \{\hat{y}\}), \tag{4}$$

where we consider the case that the degrees of $\mathsf{pa}(\hat{x})$ in R is zero and $\mathsf{pa}(\hat{y})$ in R is one; other cases are handled similarly by removing $\mathsf{pa}(x)$ and/or $\mathsf{pa}(y)$ from the above equations, which does not change the analysis below. From Lemma 5, we have $\hat{x} < \hat{y}$ since S is non-trivial. Since $\hat{x} = h(B(S)) = \min(B(S))$ and $\hat{x} < z$ for any $z \in \mathsf{children}(\hat{x})$ by definition, we see that $\hat{x} < \min(B(S) \setminus \{\hat{x}\}) = \min(B(R)) = h(B(R))$. Similarly, we also see that $\hat{y} > \max(L(S) \setminus \{\hat{y}\}) = t(L(R))$ since $\mathsf{pa}(\hat{y}) < \hat{y}$ from Lemma 1. Next, we suppose to contradict that $\hat{y} \in \mathsf{children}(\hat{x})$. Since $\hat{y} \in L(S)$, this implies that $\hat{x} \in S$ since any tree is closed upward. However, this contradicts that $\hat{x}$ is a border node in $B(S)$. Hence, $\hat{x}$ and $\hat{y}$ satisfy the pre-condition for computing $S = \mathsf{Child}_1(R)$. Finally, if we take $S' = \mathsf{Child}_1(R, x, y)$ for $x = \hat{x}$ and $y = \hat{y}$, then we have $S = S'$ from Lemma 12. This completes the proof. ■

Lemma 14 (soundness of Child_1). *Let R and S be any k-subtree of T. Suppose that S is non-serial. (1) $\mathsf{P}(S) = R$ holds if (2) $S = \mathsf{Child}_1(R, x, y)$ for some $x \in \mathsf{DelList}(R)$ and $y \in \mathsf{AddList}(R)$ such that $y \notin \mathsf{children}(x)$.*

Proof. Suppose that $S = \mathsf{Child}_1(R, x, y)$ for some $x \in \mathsf{DelList}(R)$ and $y \in \mathsf{AddList}(R)$ such that $y \notin \mathsf{children}(x)$. By definition, we have $x \in L(R)$ such that $x < h(B(R))$ and $y \in B(R)$ such that $y > t(L(R))$. By the construction of $S = \mathsf{Child}_1(R, x, y)$, we have the recurrences

$$L(S) = (L(R) \setminus \{x, \mathsf{pa}(y)\}) \cup \{\mathsf{pa}(x), y\}, \tag{5}$$
$$B(S) = (B(R) \setminus (\mathsf{children}(x) \cup \{y\})) \cup (\{x\} \cup \mathsf{children}(y)), \tag{6}$$

where we consider the case that the degrees of $\mathsf{pa}(x)$ in R is one and $\mathsf{pa}(y)$ is zero; other cases are handled similarly by removing $\mathsf{pa}(x)$ and/or $\mathsf{pa}(y)$ from the above equations, which does not change the analysis below. From Lemma 8, S is non-serial, and from Lemma 5, we have $x < y$. Clearly, $y < z$, and thus, $x < z$ for every $z \in \mathsf{children}(y)$. Thus, from the above recurrences, if $x < h(B(R))$ and $x < z$ for every $z \in \mathsf{children}(y)$, then $h(B(S)) = \min(B(S)) = x$. On the other hand, we have $\mathsf{pa}(x) < x$ from Lemma 1, and thus, $\mathsf{pa}(x) < y$. Since $y > t(L(R)) = \max(L(R))$, we have $t(L(S)) = \max(L(S)) = y$. Recall that $R' = \mathsf{P}(S) = (S \setminus \{\hat{y}\}) \cup \{\hat{x}\}$, where $\hat{x} = h(B(S))$ and $\hat{y} = t(L(S))$. Therefore, we have $\hat{x} = x$ and $\hat{y} = y$. From Lemma 12, we conclude that $R = R'$. This completes the proof. ■

From Lemma 13 and Lemma 14, we have the following theorem on the correctness of the child generation rule of type I.

Theorem 1. Let R and S be any k-subtree of T. Suppose that S is non-serial. Then, the following conditions (1) and (2) are equivalent:

(1) $\mathsf{P}(S) = R$.
(2) $S = \mathsf{Child}_1(R, x, y)$ for some $x \in \mathsf{DelList}(R)$ and $y \in \mathsf{AddList}(R)$ such that $y \notin \mathsf{children}(x)$.

Next, we have the next theorem on the correctness of the child generation rule of type II.

Theorem 2. Let R and S be any k-subtrees of T. Then, the following (1) and (2) hold.

(1) If R is pre-serial, then (i) $S = \mathsf{Child}_2(R)$ implies (ii) $R = \mathsf{P}(S)$.
(2) If S is serial, then (ii) $R = \mathsf{P}(S)$ implies (i) $S = \mathsf{Child}_2(R)$.

Proof. In general, we can show that for any set R and elements $x \neq y$ and $x' \neq y'$, if $S = (R \setminus \{x\}) \cup \{y\}$ with $x \in R, y \notin R$, and $R' = (S \setminus \{y'\}) \cup \{x'\}$ with $x \notin S, y \in S$, then $R = R'$ iff $x = x'$ and $y = y'$. (1) Suppose that $S = \mathsf{Child}_2(R) = (S \setminus \{x\}) \cup \{y\}$ with $x = \mathsf{root}(R)$ and $y = \min(\mathsf{AddList}(R))$. Now, let $x' = \mathsf{pa}(\mathsf{root}(S))$ and $y' = \mathsf{tail}(S)$. Then, clearly, $x' = x$. On the other hand, we have $y = \mathsf{tail}(R) + 1$ since parent R is pre-serial. Since this y is actually the tail of child S, we have $y' = y$. If we take $R' = \mathsf{P}(S)$, $R' = R$, and the proof is done. (2) On the contrary, suppose that $R = \mathsf{P}(S) = (S \setminus \{y'\}) \cup \{x'\}$ with $x' = \mathsf{pa}(\mathsf{root}(S))$ and $y' = \mathsf{tail}(S)$. Let $S' = \mathsf{Child}_2(R) = (S \setminus \{x\}) \cup \{y\}$ with $x = \mathsf{root}(R)$ and $y = \min(\mathsf{AddList}(R))$. Obviously, $x' = x$. Since S is serial, $\mathsf{tail}(R) = \mathsf{tail}(S) - 1 = y' - 1$. On the other hand, $y = \mathsf{tail}(R) + 1 = (y' - 1) + 1 = y'$. Hence, the theorem is proved. ■

Section 5

Theorem 4. Given an input tree T of size n, a multi-set P of colors with size m, and a positive integer $k \geq 1$, the graph motif problem for tree is solvable in $O(s+n+m)$ total time using $O(|T|)$ space.

Proof. Let $\mathcal{A}$ be a data structure for an array of $|C|$ counters $f : C \to \mathbf{N}$ with increment, decrement, and zero-test operations, where for any color $c \in C$, $inc(c)$ and $dec(c)$), respectively, increments and decrements the value of the counter $f(c)$ by one, and $zerotest()$ returns $true$ if the values of all the counter are zero and $false$ otherwise. We can easily implement such multi-counter $\mathcal{A}$ in $O(1)$ time per operation with $O(m)$ space. Initially, the algorithm set the counters by $f(c) = -f_P(c) \leq 0$ for every color c by iteratively using $dec(c)$ in $O(k)$ time. It also number the nodes of T consecutively by DFS-numbering in $O(n)$ time. Then, the algorithm enumerates all the k-subtrees of T by Algorithm1 in $O(1)$ amortized time per subset. From Theorem 3, this search requires at most $O(s)$ time in total. For each k-subset S with a pair x (or y, resp.) of deleted (or inserted, resp.) to obtain S, we increment (or decrement, resp.) the counter $f(c)$ by one, where c is the color of x (or y, resp.). Clearly, we see that $C(S)$ is identical to P as multi-set iff $f(c) = 0$ for all c that we can test by $zerotest()$ in $O(1)$ time. Hence, the result is proved. ■