PAPER Special Section on Foundations of Computer Science —New Trends in Theory of Computation and Algorithm—

Constant Time Enumeration of Subtrees with Exactly k Nodes in a Tree*

Kunihiro WASA[†a)], **Yusaku KANETA**[†b)], ***Nonmembers***, **Takeaki UNO**[††c)], ***and*** **Hiroki ARIMURA**[†d)], ***Members***

SUMMARY 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 with a tree of n nodes as an input graph, which is originally introduced by (Ferreira, Grossi, and Rizzi, ESA'11, 275–286, 2011) for general graphs. Based on reverse search technique (Avis and Fukuda, Discrete Appl. Math., vol.65, pp.21–46, 1996), we present the first constant delay enumeration algorithm that lists all k-subtrees of an input rooted tree in $O(1)$ worst-case 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.

key words: *graph algorithm, enumeration algorithm, constant delay enumeration, motif discovery, tree mining*

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], [2], [14], [15]. For instance, the proximity pattern mining problem [8], [10] 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 [10] and the graph motif problem [5], [8] 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 [6], where an instance consists of an undirected graph G of n nodes and every 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.* [6] presented the first output-sensitive algorithm that lists all k-subtrees in a graph G of size n in $O(sk)$ total time and $O(m)$ space, in other words, in $O(k)$ *amortized time* per subtree, where m 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 delay enumeration algorithm* for the k-subtree enumeration problem in *trees*. More precisely, for every $k \geq 1$, our algorithm lists all k-subtrees of an input tree T of size n in constant delay (worst-case time per subtree) using $O(n)$ preprocessing and space. Note that the delay of our algorithm is bounded by a constant independently from k and n. Our algorithm is based on reverse search technique, proposed by Avis and Fukuda [3], as in the algorithm by Ferreira *et al.* [6] for general input graphs. However, unlike their algorithm [6], our algorithm achieves the best possible enumeration complexity. Finally, we discuss an application of our algorithm to a modification of the graph motif problem for trees.

1.1 Related Work

Ferreira, Grossi, and Rizzi [6] presented an enumeration for all k-subtrees in an input graph G in $O(k)$ amortized time per solution. Ruskey [9] presented an enumeration algorithm for all subtrees in an input tree T. His algorithm runs in $O(n)$ time per solution with $O(n)$ space, where n is the number of nodes in T.

The k-subtree enumeration problem considered in this paper is closely related to a well-known graph problem of enumerating all spanning trees in an undirected graph G [12]. For this problem, Tarjan and Read [12] first presented an $O(ns+m+n)$ time and $O(m+n)$ space algorithm in 1960's, where s is the number of solutions, m is the number of edges in G, and n is the number of nodes in G. Recently, Shioura, Tamura, and Uno [11] presented $O(m+n+s)$ time and $O(m+n)$ space algorithm. Unfortunately, it is not easy to extend the algorithms for spanning tree enumeration to subtree enumeration.

One of our motivation comes from application to the graph motif problem (GMP, for short). Given a bag of k labels, called a pattern, and an input graph G, called a text, GMP asks to find a k node subgraph of G whose multi-set of labels is identical to a given pattern. Lacroix *et al.* [8] introduced the problem with application to biology and pre-

Manuscript received April 3, 2013.
Manuscript revised August 1, 2013.

[†]The authors are with the Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, 060–0814 Japan.
[††]The author is with National Institute of Informatics, Tokyo, 101–8430 Japan.

*An earlier version of this paper was presented at the 18th Annual International Computing and Combinatorics Conference (COCOON 2012).

a) E-mail: wasa@ist.hokudai.ac.jp
b) E-mail: y-kaneta@ist.hokudai.ac.jp
c) E-mail: uno@nii.ac.jp
d) E-mail: arim@ist.hokudai.ac.jp

DOI: 10.1587/transinf.E97.D.421

sented an FPT algorithm with $k = O(1)$, and NP-hardness in general. Then, Fellows *et al.* [5] showed that the problem is NP-hard even for trees of degree 3, and presented an improved FPT algorithm. Sadakane *et al.* [10] studied the string version of GMP, and presented linear-time algorithms.

Although there are increasing number of studies on GMP [5], [8], there are few attempts to apply efficient enumeration algorithms to this problem. Ferreira *et al.* [6] mentioned above is one of such studies. Recent studies [2], [14], [15] in data mining applied efficient enumeration algorithms to discovery of interesting substructures from massive structured data in the real world.

1.2 Organization of This Paper

In Sect. 2, we define basic definitions on the k-subtree problem. In Sect. 3, we first introduce the family tree for k-subtrees in a tree, and in Sect. 4, then, we present a constant delay algorithm that solves the k-subtree enumeration problem. Section 5 gives an application to the graph motif problem. Finally, in Sect. 6, we conclude.

2. Preliminaries

In this section, we give basic definitions and notation for trees and their subtrees. For the definitions not found here, please consult textbooks (see, e.g., [4]). For a set S, we denote by $|S|$ the number of elements in S. For mutually disjoint sets X and Y, we denote by $X \uplus Y$ the disjoint union of X and Y. In this paper, all graphs are simple (without self-loops or parallel edges).

2.1 Trees

A *rooted tree* is a directed connected acyclic graph $T = (V(T), E(T), \mathsf{root}(T))$, where $V = V(T)$ is a set of *nodes*, $E = E(T) \in V^2$ 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. We assume that every node v except the root has the unique parent. The *size* of T is denoted by $|T| = |V(T)| = n$. 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 $Ch(v) = \{ w \in V \,|\, (v, w) \in E \}$. We say that T is an *ordered tree* if a left-to-right order among siblings in T is given.

We define the ancestor-descendant relation $\preceq$ as follows: For any pair of nodes u and $v \in V$, if there is a sequence of nodes $(v_0 = u, v_1, \cdots, v_k = v)$ $(k \geq 0)$, where $v_0, \ldots, v_k$ are mutually distinct and $(v_{i-1}, v_i) \in E$ for every $i = 1, \ldots, k$, 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 $u \neq v$, then this relationship is denoted by $u \prec v$, and 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.

2.2 DFS-Numbering

In this paper, we regard an input rooted tree T of size $n \geq 0$ as an ordered rooted tree as follows. Given an input rooted tree T, we give an arbitrary left-to right order among siblings of T. 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 [4] 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 $u \leq v$ (resp. or $u < v$) if the numbering of u is smaller or equal to (resp. 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), (ii), and (iii):

(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 $u \not\preceq v$ and $v \not\preceq u$. For any nodes u' and v' such that $u \preceq u'$ and $v \preceq v'$, $u < v$ implies that $u' < v'$.

Proof : Properties (i) and (ii) are clear from the order of visiting nodes. For property (iii), since $u < v$, $u \not\preceq v$, and $v \not\preceq u$ hold, any nodes in $T(v)$ are visited after all nodes in $T(u)$ are visited. Furthermore, we see $u \leq u'$ and $v \leq v'$ because $u \preceq u'$ and $v \preceq v'$. Hence, $u' < v'$ holds.

2.3 K-Subtree Enumeration Problem in a Tree

Let $1 \leq k \leq n = |T|$. A *k-subset* is any subset of $V(T)$ with k nodes. A *k-subtree* in a tree T is a connected and acyclic subgraph G of T, as an undirected graph, consisting of exactly k nodes. Since T is a tree, any connected subgraph is obviously acyclic, and thus, it is completely specified by its node set. We denote by $\mathcal{S} = \mathcal{S}^{(k)}(T)$ the *family of all k-subtrees* of T.

More precisely, given a subset S of k nodes, the corresponding k-subtree $T(S)$ is specified as the subgraph $T(S) = (S, E(S))$ of T induced in S, where the edge set is given by $E(S) = \{ (u, v) \in T \,|\, u, v \in S \}$. Since $T(S)$ is connected and any connected subgraph of a tree is again a tree, $T(S)$ must be a proper k-subtree. Therefore, if it is clearly understood, we often identify a connected node set S such that $|S| = k$ with the k-subtree, where we say that S is *connected* if its induced subgraph $T(S)$ is connected. Now, we state our problem below.

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

That is, it is the problem of enumerating all elements of $\mathcal{S}$. The number of solutions is given as follows.

Lemma 2: Let $s = f(k, n)$ be the number of all k-subtrees in an input rooted tree T of n nodes.

- $s = O(n^k)$ for the upper bound.
- $s = 2^{\Omega(k)}$ for the lower bound.

Proof : For the upper bound, we can specify any k-subtree by selecting mutually distinct k nodes from n nodes of T. Thus, $s = f(k, n)$ is bounded from above by $\binom{n}{k} = O(n^k)$ for constant k. For the lower bound, we consider the infinite sequence of input rooted trees $\{T_k\}_{k\geq 1}$, where T_k is the rooted tree of height 1 and size $n = 2k - 1$ consisting of the root and $n - 1$ leaves only. Then, $s = f(k, n)$ is given by $\binom{n-1}{k-1}$. From the lower bound of binomial coefficient, this number is bounded from below

$$\binom{n-1}{k-1} \geq \left(\frac{n-1}{k-1}\right)^{k-1} = \left(\frac{2(k-1)}{k-1}\right)^{k-1} = 2^{k-1}.$$

Hence, we have $s = 2^{\Omega(k)}$. □

Our problem is a special case of the k-subtree problem in a graph, originally introduced and studied by Ferreira, Grossi, and Rizzi [6]. An input graph is a tree in our problem, while it is a general undirected graph in [6]. Ferreira *et al.* [6] showed an efficient enumeration algorithm that lists all k-subtrees in $O(k)$ amortized time per subtree for a general class of undirected graphs. However, its time complexity is still open when an input is restricted to rooted trees. Therefore, our goal is to devise an optimal algorithm that lists all k-subtrees in $O(1)$ worst-case time per subtree.

2.4 Properties of k-Subtrees

For a k-subtree S in T, we denote by $\mathsf{root}(S)$ and $L(S)$ the root and the set of leaves of S, respectively. For subset S and its complement $\bar{S} = V(T) \setminus S$, we call an edge $e = (x, y)$ of T a *cut edge* between S and $\bar{S}$ if $x \in S$ and $y \in \bar{S}$. The *border set*, denoted by $B(S)$, is the set of all lower ends y of cut edges (x, y) between S and $\bar{S}$ defined by $B(S) = \{\, y \in V(T) \,|\, (x, y) \in E(T), x \in S, y \notin S \,\} = \{\, y \in Ch(x) \,|\, x \in S, y \notin S \,\}$. In other words, $B(S)$ is the set of all nodes lying immediately outside of S. We define the *weight* of a k-subtree S by the sum $w(S) = \sum_{v\in V(S)} v \geq 0$ of the DFS numbers of the nodes in S.

Next, we introduce a class of subtrees in special form, called serial trees, as follows. For any node $r\,(1 \leq r \leq n - k + 1)$, the *serial k-subtree rooted at r* is the k-subtree $\mathcal{I}_r^{(k)}(T) = \{r, r + 1, \ldots, r + k - 1\}$ in the DFS-numbering. A k-subtree S is *serial* if $S = \mathcal{I}_r^{(k)}(T)$ for some r, and S is *non-serial* otherwise.

Lemma 3 (DFS-numbering lemma): For any k-subtree S in T, then

(i) If S is non-serial, then $\min(B(S)) < \max(L(S))$ holds, and there exists a node v that satisfies $v \in B(S)$ and $\min(S) < v < \max(S)$.
(ii) If S is serial and $B(S) \neq \emptyset$, then $\max(L(S)) < \min(B(S))$ holds.

Proof : (i) If S is non-serial, then there is some $v \in V(T)\setminus S$ such that $\min(S) < v < \max(S)$. We can find some $v' \in B(S)$ such that $\min(S) < v' < \max(S)$ and $v' \leq v$. Furthermore, if we take the smallest such v, then $v' = \min(B(S))$. Since $\max(L(S)) = \max(S)$, $\min(B(S)) < \max(L(S))$ holds. (ii) If S is serial, there is no border node between $\min(S)$ and $\max(S) = \max(L(S))$. Since any border node v is a descendant of $\mathsf{root}(S)$, we have $\mathsf{root}(S) = \min(S) \leq v$. Thus, v is properly larger than $\max(L(S))$. □

2.5 Enumeration Algorithms

We introduce terminology for enumeration algorithms according to Goldberg [7] and Uno [13]. An *enumeration algorithm* for an enumeration problem Π is an algorithm $\mathcal{A}$ that receives an *instance* I and outputs all *solutions* S in the answer set $S(I)$ into a write-only output stream O without duplicates. Let $N = \|I\|$ and $M = |S(I)|$ be the input and the output size on I, respectively. We say that $\mathcal{A}$ is of *amortized constant time* if the total running time of $\mathcal{A}$ for computing all solutions on I is linear in M. For a polynomial $p(\cdot)$, $\mathcal{A}$ is of *constant delay* using preprocessing $p(N)$ if the *delay*, which is the maximum computation time between two consecutive outputs, is bounded by a constant $c(N)$ after preprocessing in $p(N)$ time. As a computation model, we adopt the usual RAM [4].

3. The Parent-Child Relationship among k-Subtrees

Let us fix an input rooted tree $T = (V(T) = \{1, \ldots, n\}, E(T), \mathsf{root}(T) = 1)$ with size n. We assume that nodes of T are numbered by the DFS-ordering. Let $1 \leq k \leq n$ be any positive integer. In what follows, we write $\mathcal{S}$ or $\mathcal{S}^{(k)}$ for $\mathcal{S}^{(k)}(T)$ by omitting k or T. Similarly, we write $\mathcal{P}$, $\mathcal{I}$, and so on.

3.1 Basic Idea: A Family Tree

Our algorithm is designed based on *reverse search* technique by Avis and Fukuda [3]. In the reverse search technique, we define a tree-shaped search route on solutions, called a *family tree*.

A family tree for the class $\mathcal{S}^{(k)}(T)$ is a spanning tree $\mathcal{F}^{(k)}(T) = (\mathcal{S}^{(k)}(T), \mathcal{P}^{(k)}, \mathcal{I}^{(k)}(T))$ over $\mathcal{S} = \mathcal{S}^{(k)}(T)$ as node set. Given a family tree $\mathcal{F} = \mathcal{F}^{(k)}(T)$, we can enumerate all solutions using backtracking starting from the root $\mathcal{I} = \mathcal{I}^{(k)}(T)$. The collection of *reverse edges* is given by a function $\mathcal{P}^{(k)} : \mathcal{S} \setminus \{\mathcal{I}\} \to \mathcal{S}$, called the *parent function*, that assigns the unique parent $\mathcal{P}(S) = \mathcal{P}^{(k)}(S)$ to each child k-subtree S except $\mathcal{I}$. Precise definitions of $\mathcal{I}$ and $\mathcal{P}$ will be given later.

Example 1: In Fig. 1, we show an example of a family tree for all of nineteen k-subtrees of an input rooted tree T_1 of size $n = 11$, where $k = 4$.

A basic idea of the construction of the family tree $\mathcal{F}$ is explained as follows. Recall that $V(T) = \{1, \ldots, n\}$. First, we partition the family $\mathcal{S}$ of all k-subtrees in T into the mutually disjoint sub-families $\mathcal{S} = \mathcal{S}_1^{(k)}(T) \uplus \cdots \uplus \mathcal{S}_n^{(k)}(T)$, where

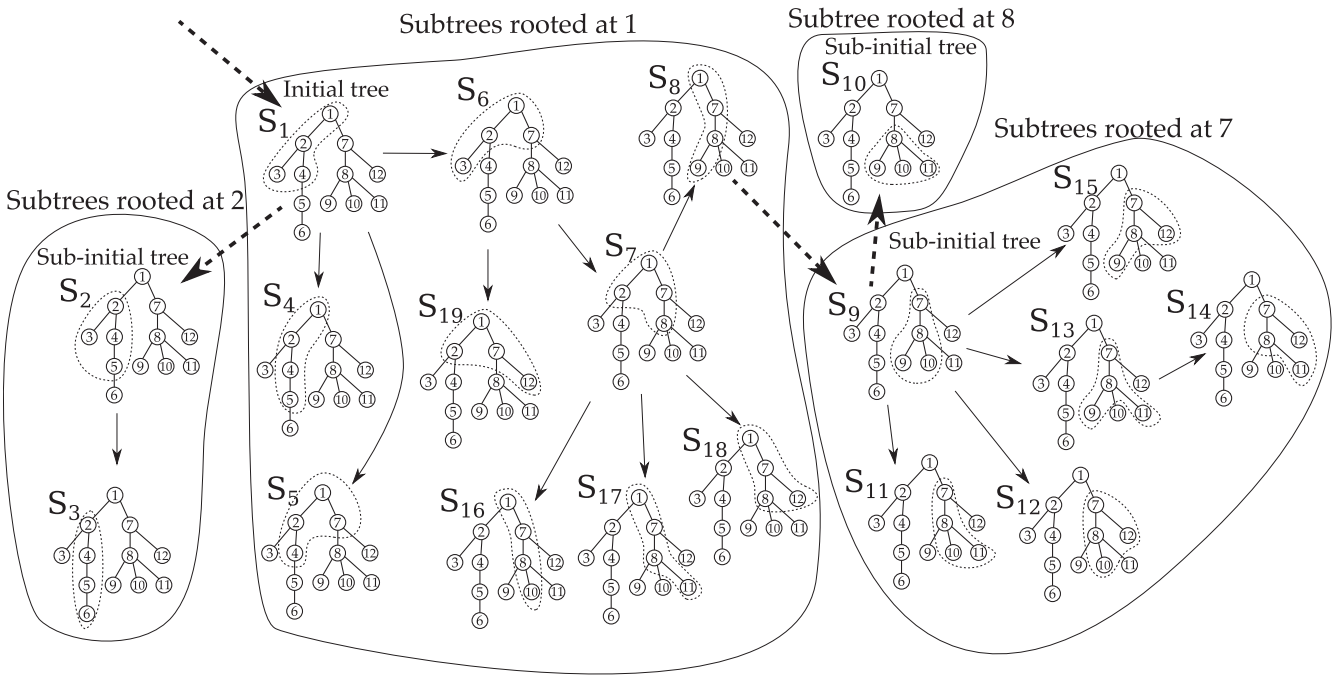

Fig. 1 A family tree for all of nineteen k-subtrees of an input rooted tree T_1 of size $n = 11$, where $k = 4$. In this figure, each set of nodes surrounded by a dotted circle indicates a k-subtree, and each arrow (resp. dashed arrow) indicates the parent-child relation defined by the parent function $\mathcal{P}_r$ for $r \in V(T_1)$ of type I (resp. $\mathcal{P}_*$ of type II). We observe that the arrows among each set of subtrees in a large circle indicates an intra-family tree, and the dotted arrows among the set of the serial and pre-serial k-subtrees form the unique inter-family tree.

for every $r \in V(T)$, the *sub-family* $\mathcal{S}_r = \mathcal{S}_r^{(k)}(T)$ is the set of all k-subtrees in T rooted at r.

Let r be any node in T. The first task is to define the family tree $\mathcal{F}_r = \mathcal{F}_r^{(k)}$, called an *intra-family tree*, for the traversal of all k-subtrees rooted at r that belongs to the sub-family $\mathcal{S}_r$. The root of the intra-family tree $\mathcal{F}_r$ is the unique serial tree $\mathcal{I}_r = \mathcal{I}_r^{(k)}(T)$ in $\mathcal{S}_r$. For the construction of $\mathcal{F}_r$, we define the parent function $\mathcal{P}_r : \mathcal{S}_r \setminus \{\mathcal{I}_r\} \to \mathcal{S}_r$ that uniquely assigns the parent $\mathcal{P}_r(S) = \mathcal{P}_r^{(k)}(S)$ with properly smaller weight to each non-serial k-subtree S rooted at r, called a k-subtree of type I. The construction of $\mathcal{P}_r$ will be described in Sect. 3.3.

The second task is to define the family tree $\mathcal{F}_* = \mathcal{F}_*^{(k)}(T)$, called an *inter-family tree*, for the traversal of sub-families $\mathcal{S}_r$'s. The root of the inter-family tree $\mathcal{F}_*$ is the unique serial tree $\mathcal{I}_1$ in $\mathcal{S}$ that has the smallest weight among all k-subtrees. We define the parent function $\mathcal{P}_* : \{\mathcal{I}_r\}_{r \in V(T)} \setminus \mathcal{I}_1 \to \mathcal{S}$ that uniquely assigns the parent $\mathcal{P}_*(S) = \mathcal{P}_*^{(k)}(S)$ with properly smaller weight to each $\mathcal{I}_r$, that is, the serial k-subtree S rooted at r. The construction of $\mathcal{P}_*$ will be described in Sect. 3.4.

Finally, we have the family tree $\mathcal{F} = (\mathcal{S}, \mathcal{P}, \mathcal{I})$ for the whole family $\mathcal{S}$ by merging all intra-family trees and the inter-family tree, where the parent function $\mathcal{P}$ is the disjoint union $\mathcal{P}_* \uplus \biguplus_r \mathcal{P}_r$, and the initial tree $\mathcal{I}$ is the unique serial tree $\mathcal{I}_1^{(k)}$ with the smallest weight. In the following sections, we will describe the details of the above construction.

3.2 Traversing k-Subtrees

We efficiently traverse between two k-subtrees R and $S \in \mathcal{S}$. Suppose we are to visit S from R. Then, we first delete a leaf $\ell \in L(R)$ from R, and next, add a border node $\beta \in B(R)$ to R. Unfortunately, this construction is not always sound, meaning that, sometimes, a certain combination of ℓ and β violates the connectivity condition on S. The next technical lemma precisely describes when this degenerate case happens and how to avoid it.

Lemma 4 (connectivity): Let R be any k-subtree of T with size $k \geq 2$. Suppose that $\ell \in R$ and $\beta \notin R$ are any nodes of $T(\mathsf{root}(R))$. Then, (i) and (ii) are equivalent:

(i) The set $S = (R \setminus \{\ell\}) \cup \{\beta\}$ is k-subtree.
(ii) $\ell \in L(R)$, $\beta \in B(R)$, and $\beta \notin Ch(\ell)$.

Proof : (i) $\Rightarrow$ (ii): By contradiction, we suppose that condition (ii) does not hold. Then, there are three cases below. (Case 1) $\ell \notin L(R)$: There dose not exist any node except ℓ such that the node is the parent of children of ℓ in T. Thus, S is not connected. (Case 2) $\beta \notin B(R)$: S consists of two or more connected sets such that one is $\{\beta\}$ and another is a set including the root node. (Case 3) $\beta \in Ch(\ell)$: See Fig. 2 for example. There is a parent-child relationship between β and ℓ in S, R, and T. Thus, S is obviously unconnected and is not a k-subtree. (ii) $\Rightarrow$ (i): $S' = R \setminus \{\ell\}$ is obviously connected. Furthermore, $S = S' \cup \{\beta\}$ is connected since

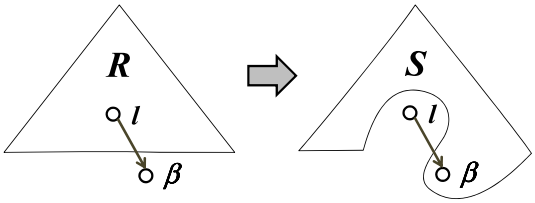

Fig. 2 The bad case for Case 3 $\beta \in Ch(\ell)$ in the proof for the connectivity lemma (Lemma 4).

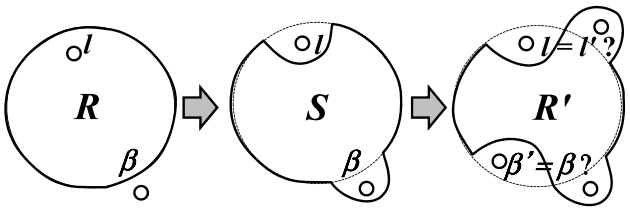

Fig. 3 An idea of the proof for Lemma 5.

$\mathsf{pa}(\beta) \in S'$. Thus, S is a k-subtree. □

The next technical lemma is useful in showing the identity of two k-subtrees.

Lemma 5 (identity): Let R be any k-subset of $V(T)$. Suppose that we take two k-subsets S and R' such that $S = (R \setminus \{\ell\}) \cup \{\beta\}$ and $R' = (S \setminus \{\beta'\}) \cup \{\ell'\}$ where $\ell \in R, \beta \notin R, \ell' \notin S$, and $\beta' \in S$. Then, we have the equivalence that (i) $R = R'$ holds iff (ii) $\ell = \ell'$ and $\beta = \beta'$ hold.

Proof : First, (ii) ⇒ (i) is obvious (See Fig. 3). Therefore, we consider (i) ⇒ (ii). Suppose that (i) $R = R'$ holds. We assume that (ii) does not hold. There are two cases below. (Case 1) $\ell \neq \ell'$: In this case, R contains ℓ but not ℓ', while R' contains ℓ' but not ℓ. Thus, R and R' can not be identical. (Case 2) $\beta \neq \beta'$: By symmetricity, R and R' can not be identical, too. By contradiction, (ii) holds. □

3.3 An Intra-Family Tree for Non-serial k-Subtrees

Firstly, for each node r in T, we describe how to build the intra-family tree $\mathcal{F}_r$ for the subspace $\mathcal{S}_r$ of all r-rooted k-subtrees of type I. Suppose that $|T(r)| \geq k$. Then, the intra-family tree $\mathcal{F}_r = (\mathcal{S}_r, \mathcal{P}_r, \mathcal{I}_r)$ is given as follows. The node set is the collection $\mathcal{S}_r$. The *sub-initial* k-subtree $\mathcal{I}_r$ is given as a serial tree containing r as its root. Actually, such a serial k-subtree is uniquely determined by the k-subtree $\mathcal{I}_r$ consisting of k nodes $\{ r+i \mid i = 0, \ldots, k-1 \}$. Next, we give the parent function $\mathcal{P}_r$ from $\mathcal{S}_r \setminus \{\mathcal{I}_r\}$ to $\mathcal{S}_r$ as follows.

Definition 1 (the parent of k-subtree of type I): Let T be an input rooted tree and $S \in \mathcal{S}_r \setminus \{\mathcal{I}_r\}$ be any non-serial k-subtree rooted at $r \in V(T)$. Then, the *parent* of S is the k-subtree

$$\mathcal{P}_r(S) = (S \setminus \{\ell\}) \cup \{\beta\} \tag{1}$$

obtained from S by deleting a node $\ell \in L(S)$ and adding a node $\beta \in B(S)$ satisfying the conditions that $\ell = \max(L(S))$ and $\beta = \min(B(S))$. Then, we say that S is a type-I child of $\mathcal{P}_r(S)$.

Lemma 6: If $S \in \mathcal{S}_r \setminus \{\mathcal{I}_r\}$, then $\mathcal{P}_r(S)$ is uniquely determined, and an well-defined k-subtree of T. Furthermore, $w(\mathcal{P}_r(S)) < w(S)$ holds.

Proof : Since S is non-serial, $\beta < \ell$ from Lemma 3. Then, we have $\beta \notin Ch(\ell)$ because if we assume that $\beta \in Ch(\ell)$ then $\ell < \beta$ from Lemma 1, and thus the contradiction is derived. It immediately follows from Lemma 4 that $\mathcal{P}_r(S)$ is connected. Since $\beta < \ell$ again, we have $w(\mathcal{P}_r(S)) = w(S) - \ell + \beta < w(S)$. □

From Lemma 6, it is natural to have $\mathcal{I}_r$ as the sub-initial k-subtree of $\mathcal{S}_r$.

Example 2: In Fig. 1, we observe that subtree S_6 is the parent of S_7 of type I since the maximum leaf is $\ell = 8$ and the minimum border node is $\beta = 3$, where $L(S_7) = \{2, 8\}$ and $B(S_7) = \{3, 4, 9, 10, 11, 12\}$.

3.4 An Inter-Family Tree for Serial k-Subtrees

To enumerate the whole $\mathcal{S}$, it is sufficient to compute the r-rooted serial k-subtree $\mathcal{I}_r$ for each possible node r in T, and then to enumerate $\mathcal{S}_r$ starting from $\mathcal{I}_r$. We see, however, that this approach is difficult to implement in constant delay because it is impossible to compute $\mathcal{I}_r$ from scratch in the constant time.

To overcome this difficulty, we define the family tree $\mathcal{F}_*$. Then, we traverse between $\mathcal{S}_r$'s using the parent function $\mathcal{P}_*$. The family tree is given by $\mathcal{F}_* = (\mathcal{S}_*, \mathcal{P}_*, \mathcal{I}_*)$, where $\mathcal{S}_*$ is the collection of serial k-subtrees and pre-serial k-subtrees, $\mathcal{I}_* = \mathcal{I}_1$ is the unique serial k-subtree with root 1, and $\mathcal{P}_*$ is the parent function from $\{\mathcal{I}_r\}_{r \in V(T)} \setminus \{\mathcal{I}_1\}$ to $\mathcal{S}$. The definition of a pre-serial k-subtree is given in Sect. 4.2. We define function $\mathcal{P}_*$ as follows.

Definition 2 (the parent of k-subtree of type II): Let S be any serial k-subtree other than $\mathcal{I}_*$. Then, the *parent* of S is the k-subtree

$$\mathcal{P}_*(S) = (S \setminus \{\ell\}) \cup \{\beta\} \tag{2}$$

obtained from S by deleting the node $\ell = \max(L(S))$ and adding the node $\beta = \mathsf{pa}(\mathsf{root}(S))$. Then, we say that S is a type-II child of $\mathcal{P}_*(S)$.

Lemma 7: If $S \in \mathcal{S}_* \setminus \{\mathcal{I}_*\}$, then $\mathcal{P}_*(S)$ is uniquely determined, and an well-defined k-subtree of T. Furthermore, $w(\mathcal{P}_*(S)) < w(S)$ holds.

Proof : If S is not the initial k-subtree $\mathcal{I}_* = \mathcal{I}_1$, β is always defined. Since ℓ is a leaf in S, $S' = (S \setminus \{\ell\})$ is obviously connected. Since β is adjacent to $\mathsf{root}(S)$, clearly, $\mathcal{P}_*(S) = (S' \cup \{\beta\})$ is also connected. Since $\beta < v$ for any node v in S, $w(\mathcal{P}_*(S)) = w(S) - \ell + \beta < w(S)$ holds. □

Example 3: In Fig. 1, we observe that subtree S_8 is the parent of S_9 of type II since the maximum leaf is $\ell = 10$ and the parent of the root is $\beta = 1$, where $L(S_9) = \{9, 10\}$ and $B(S_9) = \{11, 12\}$.

3.5 Putting Them Together

Recall that $\mathcal{S} = \mathcal{S}_1 \uplus \cdots \uplus \mathcal{S}_n$. Let $\mathcal{P}_r$ and $\mathcal{P}_*$ be the parent functions for non-serial trees for every node r in T and serial trees defined in Sects. 3.3 and 3.4, respectively. Now, we define the *master family tree* $\mathcal{F}$ for the class $\mathcal{S}$ of all k-subtrees in T by

$$\mathcal{F} = (\mathcal{S}, \mathcal{P}, \mathcal{I}), \tag{3}$$

where $\mathcal{P} : \mathcal{S} \setminus \mathcal{I} \to \mathcal{S}$ is the disjoint union $\mathcal{P}_* \uplus \biguplus_{r \in V(T)} \mathcal{P}_r$, and $\mathcal{I} = \mathcal{I}_1$ is the initial k-subtree for T. By definition, $w(\mathcal{I}) = \frac{1}{2}k(k+1)$. Furthermore, it is not hard to see that $w(S) \geq w(\mathcal{I})$ for any k-subtree $S \subseteq V$.

Theorem 1: The master family tree $\mathcal{F}$ forms a spanning tree over $\mathcal{S}$.

Proof : Suppose that starting from any $S \in \mathcal{S}$, we are to repeatedly apply the parent function $\mathcal{P}$ to S. Then, we have a sequence of k-subtrees $S_0(= S), S_1, \ldots, S_i, \ldots$, where $i \geq 0$. From Lemma 6 and Lemma 7, the corresponding properly decreasing sequence of $w(S_0) > w(S_1) > \cdots > w(S_i) > \cdots$ has at most finite length since $w(S_i) \geq 0$. Since any subtree other than the initial k-subtree $\mathcal{I}$ has the unique parent, the above sequence of k-subtrees eventually reaches $\mathcal{I}$ in finite time. □

From Theorem 1 above, we can easily show that all k-subtrees in $\mathcal{S}$ can be enumerated in polynomial delay and polynomial space by a backtracking algorithm that traverses the family tree $\mathcal{F}$ starting from the root $\mathcal{I}$.

Example 4: In Fig. 1, $\mathcal{F}^{(k)}(T_1)$ is a spanning tree on $\mathcal{S}^{(k)}(T_1)$ rooted at $\mathcal{I}^{(k)}(T_1) = S_1$ for $k = 4$. Then, we can enumerate all 4-subtrees by traversing $\mathcal{F}^{(4)}(T_1)$.

4. The Constant Delay Enumeration Algorithm

In this section, we present an efficient backtracking algorithm that enumerates all k-subtrees of an input rooted tree T in $O(1)$ delay using $O(n)$ preprocessing. The remaining task is to invert the reverse edges in $\mathcal{P}$ to compute the children from a given parent. We describe this process according to the types of a child S.

4.1 Generation of Non-serial k-Subtrees

We first consider the case that a child S is non-serial (*type I*). In our algorithm, we keep these nodes as pointers to nodes in the implementation.

Definition 3: We define the candidate sets $\mathsf{DelList}(R)$ and $\mathsf{AddList}(R)$ for deleting nodes ℓ and adding nodes β, respectively, as follows: $\mathsf{DelList}(R) = \{\, \ell \in L(R) \mid \ell < \min(B(R)) \,\}$, $\mathsf{AddList}(R) = \{\, \beta \in B(R) \mid \beta > \max(L(R)) \,\}$.

Definition 4 (child generation of type I): Given an r-rooted k-subtree R in T, we define the k-subtree

$$\mathsf{Child}_r(R, \ell, \beta) = (R \setminus \{\ell\}) \cup \{\beta\} \tag{4}$$

for (i) any $\ell \in \mathsf{DelList}(R)$ and (ii) any $\beta \in \mathsf{AddList}(R)$ such that (iii) β is not a child of ℓ.

Lemma 8 (update of lists): Let R be any r-rooted k-subtree and $S = \mathsf{Child}_r(R, \ell, \beta)$ be defined for a leaf $\ell \in \mathsf{DelList}(R)$ is removed from R and a border node $\beta \in \mathsf{AddList}(R)$ is added to R. Then,

(i) The leaf ℓ becomes the minimum border node in S.
(ii) The border node β becomes the maximum leaf in S.

Proof : From Lemma 3, we have that $\ell < \min(B(S)) < \max(L(S)) < \beta$. Hence, conditions (i) and (ii) immediately follows. □

The above lemma describes what happens when we apply Child_r to R. Now, we show the correctness of Child_r as follows.

Theorem 2 (correctness of Child_r): Let R and S be any r-rooted k-subtree of T, and S be non-serial. Then, (1) $R = \mathcal{P}_r(S)$, iff (2) $S = \mathsf{Child}_r(R, \ell, \beta)$ for (i) some $\ell \in \mathsf{DelList}(R)$ and (ii) some $\beta \in \mathsf{AddList}(R)$ such that (iii) $\beta \notin Ch(\ell)$.

Proof : Firstly, we can easily obtain a statement that $\mathsf{Child}_r(R, \ell, \beta)$ is non-serial from Lemma 3 and Lemma 4. (1) $\Rightarrow$ (2): Suppose that $R = \mathcal{P}_r(S)$. Then, R is obtained from S by removing $\ell_* = \max(L(S))$ and adding $\beta_* = \min(B(S))$. From Lemma 3, $\beta_* < \ell_*$. From Lemma 4, $\beta_* \notin Ch(\ell_*)$. Furthermore, any node $v \in S \setminus \{\ell_*\}$ is smaller than ℓ_* and any node $u \in (B(S) \setminus \{\beta_*\}) \cup Ch(\beta_*)$ is larger than β_*. Then, we see that $\max(L(R)) < \ell_*$ and $\beta_* < \min(B(R))$. If we put $\beta = \ell_*$ and $\ell = \beta_*$, then we can show that β and ℓ are a border node and a leaf in R, respectively, that satisfy the pre-condition of Child_r in Definition 4. Therefore, we can apply $\mathsf{Child}_r(R, \ell, \beta)$, and then, we obtain the new child from R by removing $\ell = \beta_*$ from R and adding $\beta = \ell_*$ to R. From Lemma 5, the child is identical to the original subtree S. (2) $\Rightarrow$ (1): In this direction, we suppose that $S = \mathsf{Child}_r(R, \ell, \beta)$ for some $\ell \in L(R)$ and $\beta \in B(R)$ satisfying the conditions (i)–(iii). Then, S is obtained from R by removing ℓ from and adding β to R. From Lemma 8, ℓ becomes $\min(B(S))$ and β becomes $\max(L(S))$. Thus, if we put $\beta_* = \ell$ and $\ell_* = \beta$, then β_* and ℓ_* satisfies the pre-condition of $\mathcal{P}_r$ in Definition 1. By applying $\mathcal{P}_r$ to S, we easily see that $(S \setminus \{\ell_*\}) \cup \{\beta_*\} = R$. Hence, the result is proved. □

4.2 Generation of Serial k-Subtrees

Next, we consider the special case to generate a serial k-subtree as a child k-subtree S of a given parent k-subtree (*type II*). A k-subtree R is a *pre-serial k-subtree* if (i) $\mathsf{root}(R)$

has only one child v such that $|T(v)| \geq k$, and (ii) v satisfies that $R \setminus \{\mathsf{root}(R)\}$ is a serial $(k-1)$-subtree of T with root v.

Lemma 9: R is a pre-serial k-subtree of T iff $\mathsf{root}(R)$ has a single child v that $|T(v)| \geq k$ and the equality $\max(L(R)) = v + k - 2$ holds.

Proof : The result follows from that a pre-serial k-subtree is obtained from a serial $(k-1)$-subtree by attaching the new root as the parent of its root. □

Definition 5 (child generation of type II): For any pre-serial k-subtree R rooted at ρ, we define

$$\mathsf{Child}_*(R) = (R \setminus \{\rho\}) \cup \{\beta\}, \tag{5}$$

where $\beta = \max(L(R)) + 1 = \min(B(R \setminus \{\rho\}))$.

Theorem 3 (correctness of Child_*): Let R and S be any k-subtrees of T. Then, the following (i) and (ii) hold:

(i) If R is pre-serial, then $S = \mathsf{Child}_*(R)$ implies $R = \mathcal{P}_*(S)$.
(ii) If S is serial and S is not $\mathcal{I}$, then $R = \mathcal{P}_*(S)$ implies $S = \mathsf{Child}_*(R)$.

Proof : Suppose that ρ is the root of R, and $R' = R \setminus \{\rho\}$. From Lemma 3 and Lemma 9, if R is a pre-serial k-subtree, then we have $\min(B(R)) = \max(R) + 1$ and $\mathsf{Child}_*(R)$ is serial. (i) Suppose that $S = \mathsf{Child}_*(R)$ with deleting ρ and adding $\min(B(R'))$. Since ρ is the parent node of $\mathsf{root}(S)$ and $\min(B(R'))$ is the largest node in S, application of $\mathcal{P}_*$ to S yields R. (ii) Suppose that R is obtained from S by $\mathcal{P}_*$ with adding the parent ρ' of $\mathsf{root}(S)$ and deleting $\beta = \max(S)$. We can easily see $\rho' = \rho$. Since S is serial, we have $\max(R) = \max(S) - 1 = \beta - 1$ and then $\min(B(R')) = \max(R)+1 = (\beta-1)+1 = \beta$, where $R' = R \setminus \{\rho\}$. Thus, we obtain S if we apply Child_* to R by deleting ρ and adding $\min(B(R'))$. This completes the proof. □

4.3 The Proposed Algorithm

In Algorithm 1, we present the main procedure EnumSubTrees and the subprocedure RecSubTree that enumerates all k-subtrees in an input rooted tree T of size n in constant delay. Starting from $\mathcal{I}$, RecSubTree recursively computes all child k-subtrees from its parent by the child generation method in this section.

The procedure RecSubTree maintains the lists of nodes $\mathsf{AddList}(S)$ and $\mathsf{DelList}(S)$ so that it can efficiently find a pair of nodes ℓ in $\mathsf{DelList}(S)$ and β in $\mathsf{AddList}(S)$ at lines 15 and 16, respectively, when it generates a child of type I by calling Child_r satisfying the conditions of Def. 4. When it backtracks to the parent, we restore the update by performing the same set of operations in the reverse order. For the generation a child of type II by Child_*, we perform similar maintenance. For keeping the present values of ℓ and β, we use the *working stack* W.

In Algorithm 2, we show the procedures $\textsc{Update}_r$ and

Algorithm 1 Constant delay enumeration for all k-subtrees in a tree.

```
 1: procedure EnumSubTrees(T, k)
 2:   Input: an input rooted tree T of size n, and size k of
      subtrees (1 ≤ k ≤ n);
 3:   Output: all k-subtrees in T;
 4:   Global variable: the working stack W;
 5:   W = ∅;
 6:   Number the nodes of T by the DFS-numbering;
 7:   Compute the initial k-subtree I of the input rooted tree T;
 8:   Initialize data structure A and B;
 9:   Update the related lists and pointers;
10:   RecSubTree(I, T, k);
11: end procedure

12: procedure RecSubTree(S, T, k)
13:   Input: an r-rooted k-subtree S, an input rooted tree T, and
      size k of subtrees;
14:   Print S;
15:   for each ℓ ∈ DelList(S) do // See Sect. 4.1.
16:     for each β ∈ AddList(S) such that β ∉ Ch(ℓ) do
17:       S ← Child_r(S, ℓ, β) with calling Update_r(A, ℓ, β);
18:       RecSubTree(S, T, k);
19:       S ← P_r(S) with calling Restore_r(A);
20:     end for
21:   end for
22:   if S is a k-pre-serial tree then // See Sect. 4.2.
23:     S ← Child_*(S) with calling Update_*(A);
24:     RecSubTree(S, T, k);
25:     S ← P_*(S) with calling Restore_*(A);
26:   end if
27: end procedure
```

Algorithm 2 Update of data structures $\mathsf{DelList}(S)$ and $\mathsf{AddList}(S)$ of type I

```
 1: procedure Update_r(A, ℓ, β)
 2:   Push ℓ̂ and β̂ in the working stack W;
 3:   L2B(ℓ); B2L(β);
 4:   ℓ̂ ← β; β̂ ← ℓ;
 5: end procedure

 6: procedure Restore_r(A)
 7:   β ← ℓ̂; ℓ ← β̂;
 8:   L2B(β); B2L(ℓ);
 9:   Pop the values of ℓ̂ and β̂ from W;
10: end procedure
```

$\textsc{Restore}_r$ for maintaining $\mathsf{AddList}(S)$ and $\mathsf{DelList}(S)$ at generation of a child of type I by Child_r using $B2L$ and $L2B$. We employ the representation of T similar to the leftmost-child right-sibling representation [4], where each node v has the pointers $v.child$, $v.prevsib$, and $v.nextsib$ for the leftmost child, the previous, and next siblings, respectively.

For all parent S, we represent two node lists $\mathsf{AddList}(S)$ and $\mathsf{DelList}(S)$ by the data structure $\mathcal{A}$ with the following information:

- Doubly linked lists of nodes $L = L(S)$ and $B = B(S)$, which consist of all leaves and all border nodes of S, respectively, in the increasing DFS order. These lists are implemented by attaching with each node v of T two pointers $v.pred$ and $v.succ$ to the predecessor and

successor in a list, respectively.
- Two pointers $\hat{\ell} = \max(L(S))$ and $\hat{\beta} = \min(B(S))$ to nodes in $L(S)$ and $B(S)$, respectively.

For a list $X \in \{L, B\}$, $X.head$ and $X.tail$ denotes the head and tail of X, respectively. Then, we define $\mathsf{DelList}(S) = \{ \ell \in L(S) \,|\, \ell < \hat{\beta} \}$ and $\mathsf{AddList}(S) = \{ \beta \in B(S) \,|\, \beta > \hat{\ell} \}$. The following operations $B2L(v)$ and $L2B(v)$ are fundamental to the maintaining of the data structure $\mathcal{A}$, where v is a node, and B_0, B_1, L_0, and L_1 are possibly empty sequences of nodes, and nodes in the lists are written in the increasing order of DFS-numbering.

- $B2L(v)$: Move a node v in B to L. To implement this operation, we rewrite (i) the present list $B = (B_0 \cdot v \cdot B_1)$ to the new list $B = (B_0 \cdot Ch(v) \cdot B_1)$, and (ii) $L = (L_0 \cdot \alpha(v) \cdot L_1)$ to $L = (L_0 \cdot v \cdot L_1)$, where the associated pointers are appropriately maintained.
- $L2B(v)$: Move a node v in L to B. To implement this, we rewrite (i) $B = (B_0 \cdot Ch(v) \cdot B_1)$ to $B = (B_0 \cdot v \cdot B_1)$, and (ii) $L = (L_0 \cdot v \cdot L_1)$ to $L = (L_0 \cdot \alpha(v) \cdot L_1)$.

In the above definition of $B2L(v)$ (resp. $L2B(v)$), $\alpha(v)$ is $\mathsf{pa}(v)$ if $\mathsf{pa}(v)$ is a leaf in S before operation (resp. after operation), and an empty sequence otherwise. Note that during the above updates, we keep all the predecessor and successor pointers unchanged except those changed by the above operations. We can show that two operations $B2L$ and $L2B$ are the inverse operations each other. We represent a k-subtree S by the following data structure $\mathcal{B}$.

- A pointer to the root ρ of S.
- A doubly linked list $L = L(S)$ of leaves of S. This is shared with $\mathcal{A}$.
- Each node v in T has two pointers lmc and rmc. If v is an internal node of S, they point to the leftmost and rightmost child of v, respectively, restricted to nodes in S. Otherwise, they are NULL.

In Algorithm 3, we show the procedure for maintaining $\mathsf{AddList}(S)$ and $\mathsf{DelList}(S)$ at generation of a child of type II by Child_*. Using the next lemma derived from Lemma 9, we can decide whether S is a pre-serial k-subtree in $O(1)$ time using a pair of pointers rmc and lmc.

Algorithm 3 Update of data structures $\mathsf{DelList}(S)$ and $\mathsf{AddList}(S)$ of type II

```
1: procedure UPDATE*(A)
2:   r ← ρ.rmc; // the unique child of ρ in S
3:   β0 ← r.prevsib; β1 ← r.nextsib;
4:   Push ρ, ℓ̂, β̂, B.head, B.tail, β0, and β1 to stack W;
5:   if β0 ≠ NULL then B.head ← β0.succ;
6:   if β1 ≠ NULL then B.tail ← β1.prev;
7:   ρ ← r; β ← B.head;
8:   B2L(β);
9:   ℓ̂ ← β; β̂ ← β.next; // β.next = max(L(S)) + 1
10: end procedure

11: procedure RESTORE*(A)
12:   β ← ℓ̂;
13:   L2B(β);
14:   Restore ρ, ℓ̂, β̂, B.head, B.tail, β0, and β1 by popping W;
15: end procedure
```

Lemma 10: S is a pre-serial k-subtree with root ρ if and only if (i) $\rho.rmc = \rho.lmc =: r$, (ii) $L.tail = r + k - 2$, and (iii) $|T(r)| \geq k$.

To check condition (iii) in $O(1)$ time, we associate $|T(v)|$ to each node v of T in the preprocessing.

Lemma 11: Let R be any pre-serial k-subtree rooted at ρ, and let r be the unique child of ρ in R. Then, $B(R \setminus \{\rho\}) = \{ v \in B(R) \,|\, r.prevsib < v < r.nextsib \}$, where we define $v.prevsib = -\infty$ (resp. $v.nextsib = +\infty$) if v has no previous sibling (resp. no next sibling).

Proof : Since a node v is in $T(r)$ if and only if $r.prevsib < v < r.nextsib$, the lemma follows. □

Assuming the above representation, we show the next lemma on the time complexity of update.

Lemma 12: The data structures $\mathcal{A}$ and $\mathcal{B}$ for $\mathsf{AddList}(S)$ and $\mathsf{DelList}(S)$ can be implemented to be updated in $O(1)$ time using $O(n)$ time preprocessing on RAM.

Proof : Initialization of the data structure is done in $O(n)$ time by once traversing an input rooted tree T. At each request for update, we dynamically update the fields of the data structure $\mathcal{A}$ according to Algorithm 2 for Child_r, and Algorithm 3 for Child_*. By construction, these operations can be implemented in the claimed complexity. □

We have the main theorem of this paper. This theorem shows that we can enumerate all k-subtrees in an input rooted tree in constant delay.

Theorem 4: Given an input rooted tree T of size n, and a positive integer $k \geq 1$, algorithm 1 solves the k-subtree enumeration problem in constant worst-case time per subtree using $O(n)$ preprocessing and space.

Proof : By the construction of RECSUBTREE in Algorithm 1, we observe that each iteration of recursive call generates at least one solution. To achieve constant delay enumeration, we need a bit care to represent subtrees and to perform recursive call. From Lemma 12, each call performs constant number of update operations when it expands the current subtree to descendants. Therefore the remaining thing is to estimate the book-keeping on backtrack. This is done as follows: When a recursive procedure call is made, we apply a constant number of operations on candidate lists and record them on a stack as in Lemma 12, and when the procedure comes back to the parent subtree, we apply the inverse of the recorded operations on the lists in constant time as in Lemma 12 to reclaim the running state in constant time. To improve the $O(d)$ time output overhead with backtrack from node v of depth $d = O(n)$ to a shallow ancestor on the family tree, we use alternating output technique (see, e.g., Uno [13]) to reduce it to exactly $O(1)$ time per solution. Combining the above arguments, we proved the theorem.

□

This result improves on the straightforward application of Ferreira *et al.*'s algorithm [6] with $O(k)$ amortized time per subtree when an input is restricted to a tree.

5. Application to the Graph Motif Problem for Trees

We consider the restricted version of graph motif problem [5], [8], called *the k-graph motif problem in a tree*. Then, we present an adaptive algorithm for the problem whose running time is proportional to the number of k-subtrees.

Let $C = \{1, \ldots, \sigma\}$ ($\sigma \geq 1$) be a set of *colors*. A *multiset* X on C is a collection of possibly duplicated colors in C. Precisely, $f_X(c)$ denotes the *count* of the color c in X. The *size* of multiset X is the total count of the colors in X defined by $\|X\| = \sum_{c \in C} f_X(c)$. For multisets X and Y on C, We define $X \subseteq Y$ if $f_X(c) \leq f_Y(c)$ for every $c \in C$, and $X = Y$ if $f_X(c) = f_Y(c)$ for every $c \in C$.

Let k be any nonegative integer and $T = (V, E, \mathsf{root}(T), \xi)$ be a vertex colored, rooted tree, where $\xi(v)$ is the color of a vertex v in V. For a subset S of nodes, we denote by $\xi(S)$ the multiset of colors appearing in S. The *k-graph motif problem in a tree* is the problem of, given a vertex colored, rooted tree T and a multiset X on C with size $k = \|X\|$, called a *pattern*, to find a k-subtree $S \subseteq V$ such that $\xi(S) = X$. This problem is known to be NP-hard even if an input is restricted to trees [5].

In what follows, we denote by s the *number of all k-subtrees in a tree T*. From Theorem 4, we have the following result.

Theorem 5: Let $k \geq 1$. Given an input rooted tree T of size n and a multiset X on C with size k, the k-graph motif problem in a tree is solvable in $O(s + n + k)$ total time using $O(n + k)$ space.

Proof : We give the algorithm Adaptive that solves the problem in the claimed complexity as follows: The algorithm uses a counter array g on c, where the counter value $g[c]$ can take either a positive, zero, or negative integer. Given a pattern X, it first initializes the counter array by $g[c] \leftarrow -f_X(c)$ for each $c \in C$ in $O(\sigma)$ time. It also initializes EnumSubTrees of Algorithm 1 in $O(n)$ time. Then, the algorithm enumerates all the k-subtrees S of T in $O(1)$ time per subtree. During the enumeration by EnumSubTrees, the algorithm increments (decrements, resp.) the counter $g[c]$ by one whenever a node labeled with color c is added to (is deleted from, resp.) S. This update of the counter can be done in constant time per subtree. When all counter values equal zero after the update, it outputs S as an output. The total running time of the algorithm is $O(s + n + k)$ time from Theorem 4. □

We compare the time complexity of the algorithm Adaptive in the proof of Theorem 5 above to that of the straightforward algorithm, called Naive here, using exhaustive search for all k-subsets. Naive solves our graph motif problem as follows: First, Naive chooses k-subset S of T, and next, checks whether S is a k-subtree. If so, Naive compares the multi-set of $\xi(S)$ and input pattern X. It outputs S as the position of X if $\xi(S) = X$, and No otherwise. Naive is done by applying the above procedure to all k-subsets of T.

From the upper bound in Lemma 2, Adaptive runs in time proportional to the actual number s of k-subtrees contained in an input rooted tree T, while Naive always requires $O(kn^k)$ time regardless of s. Therefore, we can say that our algorithm is indeed an adaptive algorithm so that it runs faster than Naive in the case that the number s is much smaller than n^k.

6. Conclusion

In this paper, we studied the k-subtree enumeration problem in rooted trees. As a main result, we presented an efficient algorithm. Our proposed algorithm solved this problem in constant worst-case time per k-subtree.

Acknowledgements

This research was partly supported by MEXT Grant-in-Aid for Scientific Research (A), 24240021, FY2012–2015, and Grant-in-Aid for JSPS Fellows (25·1149).

References

[1] S. Abiteboul, P. Buneman, and D. Suciu, Data on the Web: From Relations to Semistructured Data and XML, Morgan Kaufmann, 1999.
[2] T. Asai, K. Abe, S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa, "Efficient substructure discovery from large semi-structured data," Proc. SDM '02, pp.158–174, SIAM, 2002.
[3] D. Avis and K. Fukuda, "Reverse search for enumeration," Discrete Applied Mathematics, vol.65, pp.21–46, 1996.
[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, Introduction to Algorithms, 2 ed., The MIT Press, 2001.
[5] M. Fellows, G. Fertin, D. Hermelin, and S. Vialette, "Sharp tractability borderlines for finding connected motifs in vertex-colored graphs," Proc. ICALP '07, LNCS, vol.4596, pp.340–351, 2007.
[6] R. Ferreira, R. Grossi, and R. Rizzi, "Output-sensitive listing of bounded-size trees in undirected graphs," Proc. ESA '11, LNCS, vol.6942, pp.275–286, 2011.
[7] L.A. Goldberg, "Polynomial space polynomial delay algorithms for listing families of graphs," Proc. STOC '93, pp.218–225, ACM, 1993.
[8] V. Lacroix, C.G. Fernandes, and M.F. Sagot, "Motif search in graphs: Application to metabolic networks," IEEE/ACM Trans. Comput. Biol. Bioinformatics, vol.3, pp.360–368, 2006.
[9] F. Ruskey, "Listing and counting subtrees of a tree," SIAM Journal on Computing, vol.10, no.1, pp.141–150, Feb. 1981.
[10] K. Sadakane and H. Imai, "Fast algorithms for k-word proximity search," IEICE Trans. Fundamentals, vol.E84-A, no.9, pp.2311–2318, Sept. 2001.
[11] A. Shioura, A. Tamura, and T. Uno, "An optimal algorithm for scanning all spanning trees of undirected graphs," SIAM J. Comput., vol.26, no.3, pp.678–692, 1997.
[12] R.E. Tarjan and R.C. Read, "Bounds on backtrack algorithms for listing cycles, paths, and spanning trees," Networks, vol.5, no.3, pp.237–252, 1975.
[13] T. Uno, "Two general methods to reduce delay and change of enumeration algorithms," Tech. Rep. NII-2003-004E, National Institute of Informatics, 2003.

[14] T. Uno, T. Asai, Y. Uchida, and H. Arimura, "An efficient algorithm for enumerating closed patterns in transaction databases," Proc. DS '04, LNCS, vol.3245, pp.16–31, 2004.
[15] M.J. Zaki, "Efficiently mining frequent trees in a forest," Proc. KDD '02, pp.71–80, ACM, 2002.

Kunihiro Wasa received B.S. in Engineering in 2011 and M.S. in Computer Science in 2013 from Hokkaido University, Sapporo Japan. He is currently a doctoral student of Division of Computer Science, Graduate School of Information Science and Technology, Hokkaido University. His research interests include enumeration algorithms, data mining, and the design and analysis of algorithms in these fields. He is a member of JSAI.

Yusaku Kaneta received the B.S. degree in Engineering, and the M.S. and Ph.D. degrees in Computer Science from Hokkaido University in 2007, 2009 and 2012, respectively. Currently, he is working for Rakuten Institute of Technology, Rakuten, Inc. His research interests include Pattern Matching, Data Stream Processing, VLSI Design, and the design and analysis of algorithms in these fields. He is a member of Information Processing Society of Japan.

Takeaki Uno received the Ph.D. degree (Doctor of Science) from Department of Systems Science, Tokyo Institute of Technology Japan, 1998. He was an assistant professor in Department of Industrial and Management Science in Tokyo Institute of Technology from 1998 to 2001, and have been an associate professor of National Institute of Informatics Japan, from 2001. His research topic is discrete algorithms, especially enumeration algorithms, algorithms on graph classes, and data mining algorithms. On the theoretical part, he studies low degree polynomial time algorithms, and hardness proofs. In the application area, he works on the paradigm of constructing practically efficient algorithms for large scale data that are data oriented and theoretically supported. In an international frequent pattern mining competition in 2004 he won the best implementation award. He got Young Scientists' Prize of The Commendation for Science and Technology by the Minister of Education, Culture, Sports, Science and Technology in Japan, 2010.

Hiroki Arimura received the B.S. degree in 1988 in Physics, the M.S. and the Dr.Sci. degrees in 1990 and 1994 in Information Systems from Kyushu University. From 1990 to 1996, he was a research associate, a lecturer, and an associate professor in Kyushu Institute of Technology, and from 1996 to 2004, he was an associate professor in Kyushu University. Since 2006, he has been a professor of Hokkaido University. His research interests include data mining, computational learning theory, information retrieval, artificial intelligence, and algorithm theory. He is a member of ACM, IEICE, IPSJ, and JSAI.