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

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

[1] IST, Hokkaido University, Sapporo, Japan
{wasa,y-kaneta,arim}@ist.hokudai.ac.jp
[2] National Institute of Informatics, Tokyo, 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 nodes. Based on reverse search technique, we present the first constant delay enumeration algorithm that lists all k-subtrees of an input 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.

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 [1,2,13,14]. For instance, the proximity pattern mining problem [8,9] 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 [9] 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 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.* [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 $O(n)$ 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.

⋆ Presently at Rakuten Research, Tokyo, Japan.

J. Gudmundsson, J. Mestre, and T. Viglas (Eds.): COCOON 2012, LNCS 7434, pp. 347–359, 2012.

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, our algorithm lists all k-subtrees of an input tree T of size n in constant worst-case time per subtree using $O(n)$ preprocessing and space. 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

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 [11]. For this problem, Tarjan and Read [11] 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, and Uno [10] 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.

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 P. Lacroix *et al.* [8] introduced the problem with application to biology and presented 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.* [9] 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, Grossi, and Rizzi [6] mentioned above is one of such studies. Recent studies [2,13,14] 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 Sec. 2, we define basic definitions on the k-subtree problem. In Sec. 3, we first introduces a family tree, and in Sec. 4, then, we present a constant delay algorithm that solves the k-subtree enumeration problem. Sec. 5 gives an application to the graph motif problem. Finally, in Sec. 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, e.g., [4]. For a set S, we denote by $|S|$ the number of elements in S. In this paper, all graphs are simple (without self-loops or parallel edges).

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 $Ch(v) = \{\, w \in V \mid (v, w) \in E \,\}$.

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 $\pi = (v_0 = u, v_1, \cdots, v_k = v)$ $(k \geq 0)$, where $(v_{i-1}, v_i) \in E$ 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 and u is a *proper ancestor* of v, denoted by $u \prec v.$, 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.

DFS-Numbering. 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 [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 $x \not\preceq y$ and $y \not\preceq x$. For any nodes x', y' such that $x' \succeq x$ and $y' \succeq y$, $x < y$ implies that $x' < y'$.

The Family of k-Subtrees. Let $1 \leq k \leq n = |T|$. A *k-subtree* in a tree T is a connected and acyclic subgraph 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. Therefore, if it is clearly understood, we often identify a connected node set S such that $|S| = k$ with the k-subtree, where S is *connected* if its induced subgraph $T(S)$ is connected. In what follows, we denote by $\mathcal{S}_k = \mathcal{S}_k(T)$ the *family of all k-subtrees* of T.

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. The *border set* is the set $B(S) = Ch(S) \setminus S = \{\, y \in Ch(x) \mid x \in S, y \notin S \,\}$ that consists of all nodes lying immediately outside of S. $L(S)$ and $B(S)$ are anti-chain of nodes in T w.r.t. the ancestor-descendant relation $\preceq$. We define the *interior* and *exterior* of S, respectively, by $Int(S) = S \setminus L(S)$ and $Ext(S) = T(\mathsf{root}(S)) \setminus (S \cup B(S))$. We can easily see that interior, leaves, border, and exterior are mutually disjoint subsets of $T(\mathsf{root}(S))$. For a subset $A \subseteq S$, we define the *head* and the *tail* of S by the elements $\min(A)$ and $\max(A)$, respectively. We define the *weight* of a k-subtree S by the sum

$w(S) = \sum_{v \in V(S)} v$, of the DFS numbers of the nodes in S, where w is bounded from below by $w_{\min} = \frac{1}{2}k(k+1) \geq 0$.

Next, we introduce a class of subtrees in special form, called serial trees as follows. Let S be any k-subtree. Then, S is *serial* if it is *serial* in their shape, that is, whose nodes are consecutively numbered from its root r to the rightmost leaf $r + k - 1$, and thus, denoted by $\mathsf{sertree}(v, k) = \{ r + i \mid i = 0, \ldots, k-1 \}$. S is *non-serial* if it is not serial.

Lemma 2 (DFS-numbering lemma). *For any k-subtree S in T, then*

(a) If S is non-serial, then there is some $\min(S) < v < \max(S)$ such that $v \in B(S)$.
(b) If S is non-serial, then $B(S) \neq \emptyset$ and $\min(B(S)) < \max(L(S))$ hold.
(c) If S is serial and $B(S) \neq \emptyset$, then $\max(L(S)) < \min(B(S))$ holds.

Proof. (a) 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' is an ancestor of v. Furthermore, if we take the smallest such v, then $v' = \min(B(S))$. (b) Since $\max(L(S)) = \max(S)$ holds, the result follows from Claim (a). (c) If S is serial, there is no border node between $\min(S)$ and $\max(S)$. Since any border node is below $\mathsf{root}(S)$, it is properly larger than $\max(S)$. ■

Enumeration Algorithms. We introduce terminology for enumeration algorithms according to Goldberg [7] and Uno [12]. 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 $n = ||I||$, $m = |S(I)|$ be the input and the output size on I. 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]. 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 [6], where an input graph is a tree. They 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 $O(1)$ worst-case time per subtree.

3 The Parent-Child Relationship among k-Subtrees

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

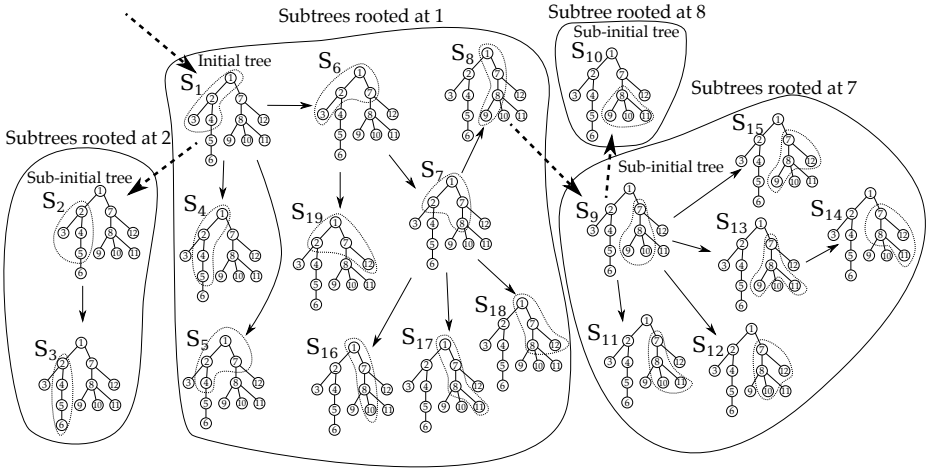

Fig. 1. A family tree for all of nineteen k-subtrees of an input tree T_1 of size $n = 11$, where $k = 4$. In this figure, each set of nodes surrounded by a dotted circle indicates a k-subtree, and each arrow (resp. dashed arrow) indicates the parent-child relation defined by the parent function $\mathcal{P}^1$ of type I (resp. $\mathcal{P}^2$ of type II). We observe that the arrows among each set of subtrees in a large circle indicates a sub-family tree of type I, and the dotted arrows among the set of the initial and sub-initial k-subtrees form the unique sub-family tree of type II.

solutions, called a *family tree*. Let $T = (V(T), E(T), \mathsf{root}(T) = 1)$ be an input tree on $V(T) = \{1, \ldots, n\}$ with DFS-numbering, and let $k \geq 1$ be any positive integer.

A family tree for the class $\mathcal{S}_k(T)$ of all k-subtrees in T is a spanning tree $\mathcal{F}_k(T) = (\mathcal{S}_k(T), \mathcal{P}_k, \mathcal{I}_k(T))$ over $\mathcal{S}_k(T)$ as node set. Given a family tree $\mathcal{F}_k$, we can enumerate all solutions using backtracking starting from the root $\mathcal{I}_k$. In what follows, we omit the subscript k and T, and thus write $\mathcal{F}$ or $\mathcal{F}(T)$ for $\mathcal{F}_k(T)$ if it is clear from context.

In the family tree, its root node is given by the unique serial tree $\mathcal{I}(T) = \mathsf{sertree}(1, k)$, called the *initial tree*, whose node set ranges from 1 to k. The collection of *reverse edges* is given by a function $\mathcal{P} : \mathcal{S}(T) \setminus \{\mathcal{I}\} \to \mathcal{S}(T)$, called the *parent function*, that assigns the unique parent $\mathcal{P}(S)$ to each child k-subtree S except the initial tree. 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 tree T_1 of size $n = 11$, where $k = 4$.

A basic idea of our algorithm is to partition the search space $\mathcal{S}(T)$ into almost mutually disjoint subspaces $\mathcal{S}(T) = (\cup_{r \in V(T)} \mathcal{S}^1(T, r)) \cup \mathcal{S}^2(T)$, where elements in collections $\mathcal{S}^1(T, r)$ and $\mathcal{S}^2(T)$ are called k-subtrees *of type I and type II*, respectively. Then, we can separately build family trees for each collections of k-subtrees.

More precisely, as we will see later, for any node r of T, called a *subroot*, the collection $\mathcal{S}^1(T, r)$ consists of all k-subtrees S whose root is r. We refer to

such k-subtrees as *r-rooted* k-subtrees. For the collection $\mathcal{S}^1(T,r)$, we can define a *sub-family tree*, denoted by $\mathcal{F}^1(T,r) = (\mathcal{S}^1(T,r), \mathcal{P}^1, \mathcal{I}^1(T,r))$, which will be given by introducing the parent function $\mathcal{P}^1$ for non-serial subtrees in Sec. 3.3. Interestingly, the initial tree $\mathcal{I}^1(T,r)$ of this collection, called the *sub-initial tree*, is naturally determined to be the serial k-subtree rooted at r by the property of $\mathcal{P}^1$ to be shown in Lemma 5 of Sec. 3.3. On the other hand, the collection $\mathcal{S}^2(T)$ of type II consists of the sub-initial trees of all collections $\cup_r \mathcal{S}^1(T,r)$ of type I, which actually are all serial k-subtrees in T. Then, the unique *sub-family tree*, denote by $\mathcal{F}^2(T) = (\mathcal{S}^2(T), \mathcal{P}^2, \mathcal{I}^2(T))$, for collection $\mathcal{S}^2(T)$ will be given by the parent function $\mathcal{P}^2$ for serial subtrees in Sec. 3.4.

3.2 Traversing k-Subtrees

We efficiently traverse between two k-subtrees, R and $S \in \mathcal{S}_k(T)$. 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 3 (connectivity). *Let R be any k-subtree R of size $k \geq 2$. Suppose that $\ell \in R$ and $\beta \notin R$ are any nodes of T that are properly below $\mathsf{root}(R)$. Then, the set $S = (R \setminus \{\ell\}) \cup \{\beta\}$ is k-subtree iff $\ell \in L(R)$, $\beta \in B(R)$, and $\beta \notin Ch(\ell)$.*

Then, the next technical lemma is useful in showing identity.

Lemma 4 (identity). *Let R and $S \subseteq V(T)$ be two k-subtrees. If we take $S = (R \setminus \{\ell\}) \cup \{\beta\}$ and $\hat{R} = (S \setminus \{\hat{\beta}\}) \cup \{\hat{\ell}\}$ for some nodes $\ell \in R, \beta \notin R, \hat{\ell} \notin S$, and $\hat{\beta} \in S$, then we have that $R = \hat{R}$ holds iff $\ell = \hat{\ell}$ and $\beta = \hat{\beta}$ hold.*

3.3 A Sub-family Tree for k-Subtrees of Type I

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

Next, we give the parent function $\mathcal{P}^1$ from $\mathcal{S}^1(T,r) \setminus \{\mathcal{I}^1\}$ to $\mathcal{S}^1(T,r)$ as follows.

Definition 1 (the parent of k-subtree of type I). Let $S \in \mathcal{S}^1(T,r) \setminus \{\mathcal{I}^1\}$ be any non-serial k-subtree rooted at $r \in V$. Then, the *parent* of S is the k-subtree $\mathcal{P}^1(S) = (S \setminus \{\ell\}) \cup \{\beta\}$ 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 $\mathcal{P}^1(S)$.

Table 1. Change of the membership of nodes w.r.t. $\mathcal{P}^1$ and Child_1 operations, where I, L, B and E are regions of k-subtree S, called interior, leaves, border, and exterior, respectively. In this figure, the leaf ℓ_* (resp. the border node β) in R corresponds to the border node β (resp. the leaf ℓ) in S.

(a) From child S to parent R via $\mathcal{P}^1$, where ℓ_* is removed and β_* is added.

node	R	S	node	R	S
$\mathsf{pa}(\beta_*)$	L	I	$\mathsf{pa}(\ell_*)$	I	L
β_*	B	L	ℓ_*	L	B
$Ch(\beta*)$	E	B	$Ch(\ell*)$	B	E

(b) From parent R to child S via Child_1, where ℓ is removed and β is added.

node	R	S	node	R	S
$\mathsf{pa}(\ell)$	I	L	$\mathsf{pa}(\beta)$	L	I
ℓ	L	B	β	B	L
$Ch(\ell)$	B	E	$Ch(\beta)$	E	B

Lemma 5. *If $S \in \mathcal{S}^1(T,r) \setminus \{\mathcal{I}^1\}$, then $\mathcal{P}^1(S)$ is uniquely determined, and a well-defined k-subtree of T. Furthermore, $w(\mathcal{P}^1(S)) < w(S)$ holds.*

Proof. Since S is non-serial, $\beta < \ell$ from Lemma 2. 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 (iii) of Lemma 3 that $\mathcal{P}^1(S)$ is connected. Since $\beta < \ell$ again, we have $w(\mathcal{P}^1(S)) = w(S) - \ell + \beta < w(S)$. ■

From Lemma 5, it is natural to have $\mathcal{I}^1(T,r)$ as the sub-initial tree of $\mathcal{S}^1(T,r)$. The next lemma describes what happens when we apply $\mathcal{P}^1$ to S.

Lemma 6 (update of lists). *Let S be any non-serial k-subtree and $R = \mathcal{P}^1(S)$. Then, (1) S and R satisfy the conditions in Table 1 (a) before and after application of $\mathcal{P}^1$. (2) The second maximum element in $L(S)$ becomes* $\max(L(R))$. *(3) The second minimum element in $B(S)$ becomes* $\min(B(R))$.

For example, 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 A Sub-family Tree for k-Subtrees of Type II

To enumerate the whole $\mathcal{S}(T)$, it is sufficient to compute the r-rooted k-serial tree $\mathcal{I}^1(T,r)$ for each possible subroot r in T, and then to enumerate $\mathcal{S}^1(T,r)$ starting from $\mathcal{I}^1(T,r)$. We see, however, that this approach is difficult to implement in constant delay because it is impossible to compute $\mathcal{I}^1(T,r)$ from scratch in the constant time. To overcome this difficulty, we want to directly build a sub-family tree $\mathcal{F}^2(T)$ of a collection $\mathcal{S}^2(T)$ of all sub-initial trees for collections $\cup_r \mathcal{S}^1(T,r)$.

The sub-family tree is given by $\mathcal{F}^2(T) = (\mathcal{S}^2(T), \mathcal{P}^2, \mathcal{I}^2)$, where $\mathcal{S}^2(T)$ is the collection, $\mathcal{I}^2(T) = \mathsf{sertree}(1,k)$ is the initial tree that is the unique serial tree with root 1, and $\mathcal{P}^2$ is the parent function from $\mathcal{S}^2(T,r) \setminus \{\mathcal{I}^2\}$ to $\mathcal{S}^2(T,r)$. Then, we define function $\mathcal{P}^2$ as follows.

Definition 2 (the parent of k-subtree of type II). Let S be any serial k-subtree other than $\mathcal{I}^2$. Then, the *parent* of S is the k-subtree $\mathcal{P}^2(S) = (S \setminus \{\ell\}) \cup \{\beta\}$ 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 $\mathcal{P}^2(S)$.

Lemma 7. *If $S \in \mathcal{S}^2(T) \setminus \{\mathcal{I}^2\}$, then $\mathcal{P}^2(S)$ is uniquely determined, and a well-defined k-subtree of T. Furthermore, $w(\mathcal{P}^2(S)) < w(S)$ holds.*

Proof. If S is not the initial tree, β is always defined. Since ℓ is a leaf of S, (i) of Lemma 3 shows that $S' = (S \setminus \{\beta\})$ is connected. Since β is adjacent to $\mathsf{root}(S)$, clearly, $\mathcal{P}^2(S) = (S' \cup \{\ell\})$ is also connected. Since $\beta < v$ for any node v in S, we have $w(\mathcal{P}^2(S)) = w(S) - \ell + \beta < w(S)$. ■

For example, in Fig. 1, we observe that subtree S_8 is the parent of S_9 of type II since the max. 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

Now, we define the *master family tree* $\mathcal{F}(T) = (\mathcal{S}_k(T), \mathcal{P}_k, \mathcal{I}_k)$, for the class $\mathcal{S}_k(T)$ of all k-subtrees in an input tree T. Let the master initial tree $\mathcal{I}_k$ be the initial tree $\mathcal{I}^2$, and let the master parent function $\mathcal{P}$ be the union of two parent functions $\mathcal{P}^1$ and $\mathcal{P}^2$ defined in the previous subsections.

Theorem 1. *The master family tree $\mathcal{F}(T)$ forms a spanning tree over $\mathcal{S}(T)$.*

Proof. Suppose that starting from any $S \in \mathcal{S}_k(T)$, we repeatedly apply the parent function $\mathcal{P}_k$ 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 5 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 $\mathcal{I}_k$ has the unique parent, the above sequence of k-subtrees eventually reaches the master subtree $\mathcal{I}_k$ in finite time. ■

For example, in Fig. 1, we see that the family tree $\mathcal{F}(T_1)$ is a spanning tree on $\mathcal{S}(T_1)$ rooted at $\mathcal{I}_k = S_1$. From Theorem 1 above, we can enumerate all k-subtrees in $\mathcal{S}_k$ starting from S_1 by depth-first search on $\mathcal{F}_k$ using backtracking.

4 The Constant Delay Enumeration Algorithm

In this section, we present an efficient backtracking algorithm that enumerates all k-subtrees of an input 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: $\mathsf{DelList}(R) = \{\ell \in L(R) \mid \ell < \max(B(R))\}$, $\mathsf{AddList}(R) = \{\beta \in B(R) \mid \beta > \min(L(R))\}$.

Definition 4 (child generation of type I). Given a k-subtree R in T, we define the k-subtree $\mathsf{Child}_1(R, \ell, \beta) = (R \setminus \{\ell\}) \cup \{\beta\}$ 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.

The next lemma describes what happens when we apply Child_1 to R.

Lemma 8 (update of lists). *Let R be any k-subtree and $S = \mathsf{Child}_1(R, \ell, \beta)$ be defined, where a leaf $\ell \in \mathsf{DelList}(R)$ is removed from and a border node $\beta \in \mathsf{AddList}(R)$ is added to R. Then, (1) R and S satisfy the conditions in Table 1 (b) before and after application of Child_1. (2) The leaf ℓ becomes the minimum border node in S. (3) The border node β becomes the maximum leaf in S.*

Now, we show the correctness of $\mathsf{Child}_1(\cdot)$ as follows.

Theorem 2 (correctness of Child_1). *Let R and S be any k-subtree of T and S be non-serial. Then, (1) $R = \mathcal{P}^1(S)$, if and only if (2) $S = \mathsf{Child}_1(R, \ell, \beta)$ 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 $\mathsf{Child}_1(R, \ell, \beta)$ is non-serial from Lemma 2 and Lemma 3. (1) $\Rightarrow$ (2): Suppose that $R = \mathcal{P}^1(S)$. Then, R is obtained from S by removing $\ell_* = \max(L(S))$ and $\beta_* = \min(B(S))$. From Lemma 6, 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_1 in Def. 4. Therefore, we can apply $\mathsf{Child}_1(R, \ell, \beta)$, and then, we obtain the new child from R by removing $\ell = \beta_*$ and adding $\beta = \ell_*$ from R. From Lemma 4, the child is identical to the original subtree S. (2) $\Rightarrow$ (1): In this direction, we suppose that $S = \mathsf{Child}_1(R, \ell, \beta)$ 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 $\min(B(R))$ and β becomes $\max(L(R))$. Thus, if we put $\beta_* = \ell$ and $\ell_* = \beta$, then β_* and ℓ_* satisfies the pre-condition of $\mathcal{P}^1$ in Def. 1. By applying $\mathsf{Child}_1(R, \ell, \beta)$, we obtain S from R by removing $\ell_* = \beta$ from and adding $\beta_* = \ell$ to R. From Lemma 4, the child is identical to the original subtree S. Hence, the result is proved. ■

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

Lemma 9. *R is a pre-serial k-subtree of T if and only if $\mathsf{root}(R)$ has a single child v and the equality $\max(L(R)) = \mathsf{root}(R) + k - 1 = v + k - 2$ holds. Furthermore, we can check this condition in constant time.*

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. ■

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

```
1: procedure EnumSubTreeMain(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:     Update the related lists and pointers;
6:     EnumSubTreeRec(I_k, T, k);

7: procedure EnumSubTreeRec(S, T, k)
8:     Print S;
9:     for each ℓ ∈ DelList(S) do // See Sec. 4.1.
10:        for each β ∈ AddList(S) such that β ∉ Ch(max(L(S))) do
11:            S ← Child_1(S, ℓ, β) by updating the related lists and pointers;
12:            EnumSubTreeRec(S, T, k);
13:            S ← P^1(S) by restoring the related lists and pointers;
14:    if S is a k-pre-serial tree then // See Sec. 4.2.
15:        S ← Child_2(S) by updating the related lists and pointers;
16:        EnumSubTreeRec(S, T, k);
17:        S ← P^2(S) by restoring the related lists and pointers;
```

Definition 5 (child generation of type II). For any pre-serial k-subtree R, we define $S = \mathsf{Child}_2(R) = (R \backslash \{\mathsf{root}(R)\}) \cup \{\beta_*(R)\}$, where $\beta_*(R) = \min(B(R))$.

Theorem 3 (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) $R = \mathcal{P}^2(S)$.
(2) If S is serial, then (ii) $R = \mathcal{P}^2(S)$ implies (i) $S = \mathsf{Child}_2(R)$.

Proof. From Lemma 2 and Lemma 9, if R is a pre-serial k-subtree, then we have $\beta_*(R) = \max(R) + 1$ and $\mathsf{Child}_2(R)$ is serial. (1) Suppose that $S = \mathsf{Child}_2(R)$ with deleting $r = \mathsf{root}(R)$ and adding $\beta_*(R)$. Since r is the root of $\mathsf{root}(S)$ and $\beta_*(R)$ is the largest node in S, application of $\mathcal{P}^2$ to S yields R. (2) Suppose that R is obtained from S by $\mathcal{P}^2$ with adding the parent r' of $\mathsf{root}(S)$ and deleting $\beta = \max(S)$. Since S is serial, we have $\max(R) = \max(S) - 1 = \beta - 1$ and then $\beta_*(R) = \max(R) + 1 = (\beta - 1) + 1 = \beta$. Thus, we obtain S if we apply $\mathsf{Child}_2(\cdot)$ to R by deleting $\mathsf{root}(R)$ and adding $\beta_*(R)$. This completes the proof. ■

4.3 The Proposed Algorithm

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

To efficiently find deleting nodes ℓ in $\mathsf{DelList}(R)$ and adding nodes β in $\mathsf{AddList}(R)$ (resp. the parent of $\mathsf{root}(R)$) by Child_1 (resp. Child_2) that satisfy

the conditions (i)–(iii) of Def. 4 (resp. Def. 5), EnumSubTreeRec maintains the lists of nodes $L(R)$ and $B(R)$ in the increasing order of DFS-numbering during the computation. This is done by attaching two pointers $\texttt{prev}_*$ and $\texttt{next}_*$ to each node v in T for implementing doubly-linked lists in addition to the standard pointers in the leftmost child, the rightmost child and right sibling representation of trees [4]. The algorithm also has two pointers $\hat{\ell}$ and $\hat{\beta}$. The pointer $\hat{\ell}$ always points to the maximum leaf of $\mathsf{DelList}(R)$ and the pointer $\hat{\beta}$ points to the minimum border node of $\mathsf{AddList}(R)$, according to Table 1 (b).

Lemma 10 (time complexity of update). *Assuming the above representation, a data structure for the above lists and pointers can be implemented to run in $O(1)$ worst case time per update using $O(n)$ time preprocessing on RAM.*

Proof. Initialization of the data structure is done in $O(n)$ time by once traversing an input tree T. At each request for update, we dynamically redirect pointers $\texttt{prev}_*$ and $\texttt{next}_*$ when a single node or a node list is deleted or added to R to maintain the values of $L(S), B(S), \hat{\ell}$, and $\hat{\beta}$ according to Table 1 (b) of Lemma 8 in the case of Child_1. We can use a similar procedure to maintain lists and pointers in the case of Child_2. Under this assumption, these operations can be implemented in the claimed complexity. ■

We have the main theorem of this paper.

Theorem 4 (constant delay algorithm for k-subtree enumeration). *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 EnumSubTreeRec 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 10, each call performs constant number of update operations when it expand 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 10, 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 10 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, we use alternating output technique (see, e.g., Uno [12]) 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 tree.

5 Application to the Graph Motif Problem for Trees

We consider the restricted version of graph motif problem where an input graph is a tree. Let $\mathcal{C}$ be a set of colors. The *graph motif problem* is the problem of, given a vertex colored graph $G = (V, E)$ and a multi-set P of colors with total frequency of colors k, to find a k-subtree $S \subseteq V$ whose multi-set of colors $C(S)$ is identical to P. We denote by s the *number of all k-subtrees in a tree T*. From Theorem 4, we have the following result.

Theorem 5. *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. We use a histgram $C : \mathcal{C} \to \mathbf{N}$ for the frequencies of colors. Initially, the algorithm sets the counter value to be $C[c] \leftarrow (-1) \cdot P[c]$ for each color in $O(|\mathcal{C}|)$ time, and also setup Algorithm 1 in $O(n)$ time. Then, the algorithm enumerates all the k-subtrees of T by in $O(1)$ delay. For each enumerated k-subtree S with removed node ℓ (or added node β, resp.), we increment (or decrement, resp.) the counter value $C[c]$ by one according to the color c of the node. We can detect the matching of P at some S by testing if all counter values equal zero in $O(1)$ time with appropriate data structure. Hence, the result is proved. ■

We can easily show that $s = n^{\Theta(k)}$. In the worst case, our algorithm is not faster than a straightforward exhaustive search algorithm with $O(mn^k)$ total time in asymptotic sense. However, in practice, 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 worst-case time per subtree. We also discussed application to graph motif problem.

References

1. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann (1999)
2. Asai, T., Abe, K., Kawasoe, S., Arimura, H., Sakamoto, H., Arikawa, S.: Efficient substructure discovery from large semi-structured data. In: SDM 2002 (2002)
3. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Applied Mathematics 65, 21–46 (1993)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. The MIT Press (2001)
5. Fellows, M., Fertin, G., Hermelin, D., Vialette, S.: Sharp Tractability Borderlines for Finding Connected Motifs in Vertex-Colored Graphs. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 340–351. Springer, Heidelberg (2007)

6. Ferreira, R., Grossi, R., Rizzi, R.: Output-Sensitive Listing of Bounded-Size Trees in Undirected Graphs. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 275–286. Springer, Heidelberg (2011)
7. Goldberg, L.A.: Polynomial space polynomial delay algorithms for listing families of graphs. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, STOC 1993, pp. 218–225. ACM, New York (1993)
8. Lacroix, V., Fernandes, C.G., Sagot, M.-F.: Motif search in graphs: Application to metabolic networks. IEEE/ACM TCBB 3, 360–368 (2006)
9. Sadakane, K., Imai, H.: Fast algorithms for k-word proximity search. IEICE Trans. Fundam. Electron., Comm., and Comp. E84-A(9), 2311–2318 (2001)
10. Shioura, A., Tamura, A., Uno, T.: An optimal algorithm for scanning all spanning trees of undirected graphs. SIAM J. Comput. 26(3), 678–692 (1997)
11. Tarjan, R.E., Read, R.C.: Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. Networks 5(3), 237–252 (1975)
12. Uno, T.: Two general methods to reduce delay and change of enumeration algorithms. Technical Report NII-2003-004E. National Institute of Informatics (2003)
13. Uno, T., Asai, T., Uchida, Y., Arimura, H.: An Efficient Algorithm for Enumerating Closed Patterns in Transaction Databases. In: Suzuki, E., Arikawa, S. (eds.) DS 2004. LNCS (LNAI), vol. 3245, pp. 16–31. Springer, Heidelberg (2004)
14. Zaki, M.J.: Efficiently mining frequent trees in a forest. In: KDD, pp. 71–80 (2002)