Doctoral Dissertation

Constant time enumeration of weighted trees and its applications

Mengze QIAN

Supervisor Ryuhei UEHARA

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
(Information Science)

September 2025

Abstract

The enumeration problem is a fundamental issue across various disciplines, requiring the output of all elements from a specified set without duplicates or omissions. In graph theory, the enumeration problem of the graph class always restricted by the number of vertices for graphs. Since graph structures form the backbone of many domains within computer science, including algorithms, network theory, computational biology, and data modeling, the enumeration of a specific graph class is an important problem in graph theory and combinatorics. In general, enumeration algorithms are designed individually for each specific graph class. Our motivation is to present a comprehensive framework for graph enumeration that is applicable to various graph classes. To address this, we introduce a modular two-phase framework that is both expressive and widely applicable, capable of exhaustively and non-redundantly enumerating graphs under various structural constraints.

The heart of this framework lies in the block-cutpoint tree (BC-tree). Tree structures hold a central place due to their simplicity and power in representing hierarchical relationships, dependencies, and decompositions. Of particular importance are BC-trees, which are tree representations of connected undirected graphs where the nodes correspond to either blocks (biconnected components) or cutpoints (vertices whose removal increases the number of connected components).

A BC-tree reflects the global structure of a connected graph in terms of its cutpoints and blocks. Given their relevance in areas such as network analysis and graph decomposition, the ability to generate all BC-trees of a fixed size under structural constraints is an essential foundational step toward the enumeration of wider graph families. We propose the first constant time enumeration algorithm for the class of BC-trees. The key feature of this algorithm is its ability to enumerate each satisfied BC-tree exactly once, without duplication, and with a constant time delay per output. The algorithm operates by reverse search, which defines a parent-child relationship among a target set of BC-trees and traverses this implicit family tree of all solutions without constructing it explicitly. Theoretical analysis guarantees the algorithm's correctness and output sensitivity. Previous studies have not thoroughly studied efficient enumeration algorithms for BC-trees, despite their theoretical and practical significance.

To further enrich this framework and address the enumeration problems for wider graph classes, we introduce an enumeration algorithm for rooted

vertex-weighted trees (referred to as weighted trees). A weighted tree is a rooted tree where each vertex is assigned a non-negative integer weight. For the enumeration problem of weighted trees, the tree structure and the total weight are fixed. This model arises in practical domains such as hierarchical clustering, compressed data structures, and frequency-based modeling. In our framework, weighted trees can be used, for example, to determine the number of vertices across the blocks in a graph corresponding to a BC-tree.

The weighted tree enumeration algorithm also uses the reverse search technique and is capable of enumerating all possible weight assignments that meet a specified structure and total weight condition with a constant time delay per tree. The design of the algorithm carefully handles the partitioning of integer weights for subtrees, especially for isomorphic subtrees, while keeping parent-child relationships on weight sequences. The enumeration process avoids redundant generation and guarantees exhaustiveness. Through theoretical analysis, we show that the algorithm is efficient and scalable and that it can be smoothly integrated into the graph enumeration framework to support weighted graph generation. Our framework thus becomes a powerful and flexible tool for the generation of graph classes with specific constraints, such as the vertex number of the graph.

Then, we propose the notion of the enumeration framework. The framework consists of two phases: The first phase involves enumerating all possible BC-trees that represent the skeletons of the graphs in the target graph class. More specifically, a BC-tree captures the connectivity structure among blocks within a connected graph. This step generates the abstract structure that underlies the target graphs. In the second phase, for each BC-tree obtained, we enumerate all possible graphs belonging to the target class that correspond to it. It leverages the fact that each undirected connected graph corresponds uniquely to a BC-tree, which effectively prevents redundancy in the enumeration. In other words, the output contains no isomorphism. This phase ensures comprehensive coverage of the target graph class while maintaining efficiency.

As described above, to realize this enumeration framework, the problem can be decomposed into two subproblems: the enumeration of BC-trees and the enumeration of vertex-weighted trees corresponding to each BC-tree. The results of these two subproblems allow us to construct a unified and efficient enumeration algorithm for a wide range of graph classes. Then, we show instances for the application of the enumeration framework. In this framework, the first phase leverages the constant time BC-tree enumeration algorithm. The second phase depends on the specific graph class being targeted—such as block graphs and cactus graphs—and involves the generation of concrete block structures that satisfy class-specific constraints. For instance, in the

case of block graphs, each block must be a clique, while for cactus graphs, each block must be a cycle or an edge. The realization phase may also include conditions on cutpoint positions, block sizes, etc.

We outline the main contributions of this dissertation below. First, we present the first known constant time enumeration algorithm for BC-trees under the constraint of the number of vertices, providing a foundational tool for graph decomposition and reconstruction. Second, we propose the first constant amortized time enumeration algorithm for weighted trees with a fixed weight sum and a tree structure, and demonstrate how it enhances the expressive power and flexibility of our graph enumeration framework. Third, we propose a notion of a two-phase enumeration framework for graph classes that leverages BC-trees as structural skeletons and supports a wide variety of graph classes by varying the block realization rules.

This research significantly advances the understanding of enumeration problems in graph theory and opens new avenues for practical applications. The combination of efficient enumeration techniques and a modular framework enables the systematic study and enumeration of complex graph classes. Potential applications include network analysis (where reliability and redundancy depend on structural decomposition), bioinformatics (where molecular structures are often represented as block graphs or trees), chemistry (for enumerating feasible compound structures), and artificial intelligence (for structured model space exploration). The proposed methods also have implications for database design, compressed representation of hierarchical data, and algorithmic testing through exhaustive instance generation.

In summary, this dissertation proposes the first known constant time enumeration algorithm for BC-trees and the first known constant amortized time enumeration algorithm for weighted trees. It also proposes the notion of a brand-new framework for enumeration through weighted BC-trees, setting the stage for future research into more general and expressive enumeration frameworks.

Keywords– graph enumeration, reverse search, BC-tree, rooted vertex-weighted tree, enumeration framework

Acknowledgements

First of all, I sincerely thank my supervisor, Prof. Ryuhei Uehara, for his great support during my master's and doctoral studies. He not only gave me professional advice on studies and research but also actively encouraged me to attend research conferences and give research presentations. He served as a role model for me as a researcher.

I fully appreciate my parents and family for their understanding and persistent, unselfish support. Their support allowed me to make it through and reach where I am today.

I would also like to thank Ms. Tomoko Taniguchi, who always provided me selfless and full assistance with my English grammar. I appreciate all the individuals I have met throughout the years as well as the lab members. I'm glad to have worked with you until today, and becoming a part of the Uehara Laboratory makes me glad.

Contents

List of Figures

Chapter 1

Introduction and Backgrounds

Research on specific graph classes has always been of immense value in graph theory. Many specialized graph classes have had extensive use across multiple fields. From an overall perspective, exploring applications of graph theory, particularly for specific graph classes, shows a natural relation with the outputs of the catalogs for these classes. This observation motivates an investigation of the enumeration problem in graph classes. The enumeration problem demands the generation of all graphs within specified constraints, usually determined by the number of vertices. The enumeration ensures that there is no duplication or omission for a specified graph class. Recent years included a comprehensive study of the enumeration problem within graph classes. Yamazaki et al. proposed an enumeration algorithm for interval graphs and permutation graphs in [28]. Following that, they applied an identical framework to develop an enumeration algorithm for distance-hereditary graphs and Ptolemaic graphs in [27]. The catalogs for these graph classes have been updated on the website[1]. Nakano and Uno provided an algorithm for enumerating trees with a specific diameter in [18], showing that each tree can be enumerated with a constant time delay. All these algorithms used reverse search [2] as the enumeration technique, a widely used method, especially for enumeration problems, whereas the enumeration problem of BC-trees is still unexplored.

The tree structure, a fundamental data structure and a specific graph class in computer science, has been thoroughly studied over the past few decades. Many disciplines extensively exploit various specific tree structures. Moreover, specific tree structures are correlated with the enumeration problem. Beyer and Hedetniemi, in their work referenced as [4], proposed an algorithm that generates all rooted trees containing n vertices. This algorithm

[1] `http://www.jaist.ac.jp/~uehara/graphs/`

uses the successor function, which is based on the lexicographical order of the tree's depth sequence. Robert et al. extended upon the results of Beyer and Hedetniemi in [26] by applying them for the generation of unrooted trees. Furthermore, additional studies concentrate on tree enumeration, such as [19, 14, 15]. Nonetheless, the enumeration of weighted trees has yet to be presented.

In graph theory, a set of generation rules is essential for describing a certain class of graphs. Such a set involves local operations, each generating new vertices and edges in the graph in distinct ways, ensuring that both the original graph and the resulting graph belong to the same target set. The generation rules allow the construction of all graphs within the class based on the most fundamental element, such as a single vertex and an edge. In 1986, Bandelt and Mulder developed a set of generation rules for the class of distance-hereditary graphs [3]. Nakano et al. subsequently applied these rules to develop an efficient enumeration algorithm for the class [16]. This approach has also been applied to Ptolemaic graphs [27]. Brandstädt and Le presented a set of generation rules for 3-leaf power graphs in [5], a class of graphs relevant to computational biology. The generation rules play a role in these graph classes (see [11]).

In 1969, Harary established the notion of the block-cutpoint tree (BC-tree) for a connected graph, as referenced in [13]. In a connected graph, a vertex is called a cutpoint if its removal turns the graph disconnected, while a block of a graph is defined as a maximal nonseparable subgraph, where a nonseparable graph is biconnected and nontrivial. A BC-tree contains two distinct types of vertices: block vertices and cutpoint vertices. A block vertex represents a block, whereas a cutpoint vertex represents a cutpoint within the graph. A block vertex is adjacent to a cutpoint vertex if the associated block contains that cutpoint in the graph. The BC-tree concept has been utilized across various fields, as shown by its application in chemistry, information systems, and mathematics [20, 21, 7, 30, 29, 10, 24]. As far as the author knows, a comprehensive set of generating rules for BC-trees has not been established.

Our motivation stems from the practical application of the block-cutpoint tree (BC-tree). The BC-tree is obtained from a graph whose structure has been partially simplified. By considering the procedure in reverse, we may initially enumerate the associated BC-trees for a graph class, and subsequently, for each enumerated BC-tree, we can fill in all the missing details to enumerate the respective graph class. Thus, our primary objective is to create an efficient enumeration algorithm for the BC-tree, which is based on a set of generation rules of the BC-tree we proposed. Subsequently, we go over the possibility of a graph derived from a BC-tree. Each block ver-

tex in a BC-tree corresponds to a block in the original graph, allowing for straightforward reversion of a block vertex to a block with a certain number of vertices based on its vertex number. Subsequently, we may enumerate the graph class using the BC-tree, which directs us to the second purpose of this study. We aim to enumerate every possible weighted BC-tree for a particular BC-tree and the associated total weight; a weighted BC-tree is defined as a BC-tree where each vertex is assigned a weight. This dissertation proposes the enumeration of weighted trees rather than weighted BC-trees, as a BC-tree is a particular form of tree structure. Numerous works are related to the weighted tree problem, with the characterization of the weighted tree varying significantly across different fields. The enumeration algorithm for weighted trees has yet to be proposed, as the structure is limited to a tree structure where only the vertices are assigned values.

1.1 Our Contribution

The main result of this dissertation is as follows.

1.1.1 Enumeration on BC-trees

We firstly concentrate on the enumeration problem associated with BC-trees. Here, we clear the enumeration problem of BC-trees. For the given integer n, the algorithm enumerates all BC-trees where the number of block vertices in each output is less than n. The BC-tree is regarded as a rooted tree, whereas a specific vertex is designated as the root of the tree.

At first, we propose a set of generation rules for rooted BC-trees. In a graph class, the set of generation rules involves a series of operations that induce local changes in the graph, leading to a new graph that remains within the same graph class as the original. For instance, we can generate any tree from a single vertex by repeatedly adding pendant vertices. We define the generation rules of BC-trees as a specialized tree structure to guarantee that each operation adds exactly one block vertex to the BC-tree, resulting in a newly generated tree that stays a BC-tree.

In addition, according to the generation rules, we investigated the enumeration problem of rooted BC-trees, which is divided into the enumeration of ordered BC-trees and unordered BC-trees. An ordered tree T assigns a specific order to each vertex's descendants. Thus, exchanging the positions of any two siblings in T may result in an entirely different tree. Conversely, an unordered tree T is defined as a tree in which the descendants of any vertex hold a free order. We use the reverse search technique [2] for enu-

meration. Reverse search is a useful technique, particularly in the context of enumeration problems. By defining the family tree within the target set and traversing it, every element within the target set can be enumerated without omission or duplicates.

We presented the enumeration of both ordered and unordered BC-trees by combining the generation rules for BC-trees with the reverse search technique. The enumeration of ordered BC-trees can be done by extending the framework provided in [15], allowing the enumeration of each ordered BC-tree in constant time. Additionally, we extend the framework provided in [17] to construct an enumeration algorithm for unordered BC-trees, allowing the enumeration of any unordered BC-tree in constant time, which is also simpler to implement.

1.1.2 Enumeration on weighted trees

We then move to the enumeration algorithm for weighted trees. A weighted tree, in this context, is defined as a tree in which each vertex is assigned a weight. Furthermore, we clarify the enumeration problem with weighted trees; for two given integers n and w, the algorithm enumerates all weighted trees with a vertex number of no more than n and a total vertex weight w. In this context, we consider weighted trees as rooted trees.

The enumeration of weighted trees has two steps. At first, it enumerates all rooted trees containing no more than n vertices. Next, it enumerates all weighted trees for each rooted tree, with a total weight of w. Numerous previous studies discuss the efficient enumeration of trees (see [23, 15, 26, 17], for example), allowing the linear time enumeration of rooted trees to be an unchallenging task. Therefore, this study focuses mainly on the second step of the algorithm. To clarify the problem, for a given ordered tree and a total weight, our algorithm enumerates all possible weight sequences for weight trees suited to such an ordered tree with the given total weight, where the weight sequence consists of all weights of the tree's vertices in pre-order traversal. Afterwards, we can enumerate all weight sequences suitable for the canonical tree with a total weight of w.

The reverse search technique is also used in the enumeration of weighted trees. Specifically, we define a unique parent-child relationship among all weight sequences that satisfy the criteria. Subsequently, the family tree can be constructed using this relationship in a straightforward way. The algorithm enumerates all weight sequences by outputting only the difference from the previous sequence and achieves this objective in constant amortized time. Afterwards, we can enumerate all weighted trees with a maximum of n vertices and a total weight of w in constant amortized time.

1.1.3 Enumeration framework for graph classes

We propose a comprehensive and general notion of an enumeration framework that systematically enumerates connected graphs using BC-tree representations. This framework is based on the structural dissection of connected graphs into their block-cutpoint trees (BC-trees), which represent the connection among blocks of a graph as a tree with nodes corresponding to either blocks (maximal biconnected subgraphs) or cutpoints (articulation vertices). The primary innovation consists in dividing the graph enumeration process into two different but interconnected phases: skeleton enumeration and graph realization.

In the beginning, we enumerate all BC-trees that match the structural constraints of the assigned graph class. This is achieved by an efficient algorithm previously proposed in the dissertation, which, using the reverse search technique , allows the constant time enumeration of rooted BC-trees. These BC-trees serve as skeleton structures, abstractly representing the connectivity and cutpoint arrangement without specifying particular graph-level specifics such as block contents.

The second phase involves completing the implementation of each enumerated graph by creating graphs that correspond with the given BC-tree. The resulting procedure should be modified for the specific type of graphs under consideration. In block graphs, each block is required to be a complete subgraph (clique), while in cactus graphs, blocks are restricted to either simple cycles or individual edges. The realization phase may additionally include supplementary constraints, such as limitations on block size, regulations on cutpoint distribution, or edge labeling methodologies, contingent upon the application.

By abstracting the enumeration process into this two-phase framework, we provide an integrated and scalable way of producing an extensive range of graph classes. The separation between the construction of general BC-trees and the inner structure of blocks allows flexible algorithm creation and reuse. This framework generalizes the enumeration of BC-trees and provides a practical, scalable method for solving enumeration problems for various graph classes in graph theory.

1.2 Significance

Abstracting sets of closely related vertices in a graph is a significant task for many different contexts. The BC-tree derived from a graph is a tree structure that shows the relationships among blocks and cutpoints within the graph,

serving as its skeleton structure. The weighted BC-tree is an extension of a BC-tree in which each vertex is assigned a weight. By extending the enumeration algorithm of weighted BC-trees to deal with the enumeration problem of more graph classes, a framework is expected to efficiently enumerate a wider variety of graph classes. For instance, block graphs can be enumerated under the same time complexity by creating a one-to-one correspondence between the set of weighted BC-trees and block graphs; additional graph classes, including cactus graphs and 3-leaf power graphs, can similarly be enumerated using the same framework.

Unlike the counting problem, the enumeration problem requires the listing of all feasible solutions to a certain problem, which is especially helpful when the objective of the problem is unclear and requires a review of every possible solution. The tree structure often serves as the fundamental data structure, deeply investigated over the past decades and widely used across various fields. Numerous varieties of tree structures exist. The weighted tree is a distinct tree structure derived from cases in which members are influenced by specific weights or costs. On the other hand, a weighted tree is defined as a tree in which its edges or vertices are assigned weights. In the applications of the weighted tree under our context, only the vertex weights are significant. In this dissertation, we define a weighted tree as a tree in which only the vertices contain weights; the total weight of the tree is the sum of the weights of all its vertices. Due to its characteristic, this class of weighted trees has naturally applied in wide areas; nevertheless, the enumeration algorithm for this class has yet to be provided.

1.3 Organization of Dissertation

The dissertation is organized as follows:

Chapter 2 gives the preliminaries of the study. The majority of the notions presented are based on the basic concepts related to graph theory, and we will give their mathematical definitions. The remaining content provides concepts relevant to this study, such as the weighted tree. The weighted tree can be characterized as edge-weighted trees, vertex-weighted trees, and trees where all edges and vertices are weighted, depending on the context; however, this paper only focuses on vertex-weighted trees.

Chapter 3 concentrated on the enumeration of block-cutpoint trees. For the given integer n, we proposed an algorithm that allows the enumeration of BC-trees containing no more than n block vertices in constant time for each tree. In our context, the block-cutpoint tree is recognized as rooted.

In Section 3.1, we first established the generation rules for BC-trees. Each

generation rule adds precisely one block vertex to a BC-tree in a specified manner, and the execution of any generation rule on a BC-tree results in a newly formed BC-tree. Therefore, all rooted BC-trees can be derived by continually using one of the generation rules from the simplest form of a BC-tree, which is either a K_1 containing a single block vertex or a K_2 containing both a block vertex and a cutpoint vertex and rooted at the cutpoint vertex.

Following that, in Section 3.2, we concentrate on the enumeration of ordered BC-trees. An ordered BC-tree is a BC-tree in which the descendants of each vertex are designated in a particular order. By combining the generation rules of BC-trees, we generate the family tree of ordered BC-trees by reverse search, and an ordered BC-tree with no more than n block vertices can be enumerated in constant time for each tree.

Next, in Section 3.3, we discuss the enumeration of unordered BC-trees. Applying the enumeration algorithm of ordered BC-trees on unordered BC-trees results in repeated outputs. Thus, we then define the canonical representation of the unordered BC-tree, and by cataloging these canonical representations, we can avoid duplicates. Likewise, an unordered BC-tree containing no more than n block vertices can be enumerated in constant time for each tree.

Next, we generalize the enumeration of weighted BC-trees to consist of the enumeration of weighted trees. Chapter 4 concentrated on the enumeration of weighted trees. We focus on the enumeration algorithm that, for given numbers n and w, enumerates all weighted trees with a maximum of n vertices and a total weight of w. The weighted tree is considered an ordered tree. We divide the task into two steps: first, we enumerate all ordered trees containing no more than n vertices; then, for each tree, we enumerate all weighted trees corresponding to that tree with a total weight of w. As the first step is quite uncomplicated, this dissertation will concentrate solely on the second step.

In Section 4.1, we proposed the concept of a reduced-set tree to enumerate weighted trees for a certain tree and total weight. A reduced-set tree arises from a tree by compressing all isomorphic subtrees rooted at a set of siblings into a singular vertex. In a reduced-set tree, two subtrees deriving from a pair of siblings must be distinct, a condition referred to as a sibling-distinct tree.

Section 4.2 commences with the enumeration of weighted sibling-distinct trees. Due to the property of the sibling-distinct tree, where no two subtrees rooted at a sibling pair are isomorphic, it is sufficient to enumerate the weighted sibling-distinct tree as a sequence of integers with specified digits and the sum of these integers. The sibling-distinct tree is considered a basic instance of tree structure. Using reverse search, the weighted sibling-distinct tree can be enumerated in constant time for each output.

Then, we generalize this result to a general tree structure in Section 4.3. By using the algorithm indicated in Section 4.1, a general tree can be converted into its equivalent sibling-distinct tree. After that, utilizing the enumeration algorithm described in Section 4.2, we can enumerate each weighted tree in constant time.

Chapter 5 presents an extensive enumeration framework for connected graph classes utilizing the structural representation provided by BC-trees. This framework extends the constant time enumeration approaches to BC-trees established in the preceding chapter by dividing the graph enumeration process into two phases: skeleton enumeration and graph realization. During the initial step, all BC-trees, with the restriction of the number of block vertices, are efficiently enumerated, functioning as abstract "skeletons" that depict the block-cutpoint structure of target graphs. In the second phase, each BC-tree is transformed into actual graphs by allocating internal structures to blocks and establishing the distribution of cutpoints among these blocks.

This chapter illustrates the adaptability and expandability of the framework by showing how to apply it to many significant graph classes, such as block graphs, cactus graphs, and 3-leaf power graphs. The system adjusts to the unique block constraints and structural characteristics of each class, demonstrating that enumeration may be dealt with via BC-tree representations. The chapter additionally presents concrete examples of graph classes to elucidate the practical execution of each phase.

Chapter 6 presents the dissertation's conclusion, followed by proposed future work relating to this result, which includes improvements to the algorithms and a discussion on the application of the enumeration of weighted trees and weighted BC-trees.

Chapter 2

Preliminaries

We first clarify that the graph in the following context is considered as an undirected finite graph.

In a graph $G = (V, E)$, a sequence of vertices $(v_0, v_1, \ldots, v_m)$ is a *path* if $\{v_i, v_{i+1}\} \in E$ for each $i = 0, 1, \ldots, m-1$. The *length* of a path is the number of edges in the path. For two vertices u and v in a graph, the *distance* of the vertices, denoted by $dist(u, v)$, is the length of a shortest path joining u and v.

For two vertices $u, v \in V$, we call that they are a pair of *twins* if $N(u)\backslash\{v\} = N(v)\backslash\{u\}$. For a pair of twins u and v, we say that they are *strong twins* if $\{u, v\} \in E$, and *weak twins* if $\{u, v\} \notin E$.

A graph $G = (V, E)$ is a *tree* if it is connected and acyclic. In a tree, a vertex is called a *leaf* if it has degree 1. It is known that any pair of vertices on a tree is joined by a unique path (see, e.g., [13]).

A *rooted* tree T is a tree with one vertex r chosen as its *root*. A subtree of T is a tree that descends from a vertex in T. The subtree rooted at vertex s is denoted by $T(s)$. The *depth of a rooted tree* is the maximum depth of a leaf and it is denoted by $d(T)$. The *depth of a vertex* v in a rooted tree is the length of the path connecting v and root r, denoted by $d(v)$. The *height* of T is the maximum depth among leaves of T, denoted as $height(T)$. In a rooted tree, two vertices are called *siblings* if two vertices are connected to the same vertex in the tree and have the same distance from the root. Two subtrees $T(s)$ and $T(s')$ in T are called *sibling subtrees* if s and s' are siblings in T.

An *ordered tree* is a rooted tree that the children of each vertex are specified in left-to-right ordering. Given an ordered tree T, let $(v_1, v_2, \ldots, v_n)$ be the sequence of the vertices of T in *preorder*, which is given by the depth-first search from the root r (see, e.g., [8] for the details). The sequence $L(T) = (d(v_1), d(v_2), \ldots, d(v_n))$ is called the *depth sequence* of T.

Let T_1 and T_2 be two ordered trees, $L(T_1) = (a_1, a_2, \ldots, a_n)$ and $L(T_2) = (b_1, b_2, \ldots, b_m)$ be the depth sequences of T_1 and T_2, respectively. If either $a_i = b_i$ for each $i = 1, \ldots, j-1$ (possibly $j = 1$) and $a_j > b_j$, or $L(T_2)$ is the prefix of $L(T_1)$, namely $a_i = b_i$ for each $i = 1, 2, \ldots, m$ and $n > m$, we say that $L(T_1)$ is *heavier* than $L(T_2)$, denoted by $L(T_1) > L(T_2)$. T_1 and T_2 are *isomorphic* if $n = m$ and $a_i = b_i$ for each $i = 1, \ldots, n$.

A *weighted tree* is a tree in which each vertex has a non-negative weight. Let T be a weighted tree and v be a vertex in T, the *weight of vertex* v is denoted by $w(v)$ where $w(v) \geq 0$. The *weight of tree* T is the sum of the weights for all vertices in the tree, denoted by $w(T) = \sum_{v \in T} w(v)$. Given a weighted ordered tree T with preorder $(v_1, v_2, \ldots, v_n)$, the sequence $W(T) = (w(v_1), w(v_2), \ldots, w(v_n))$ is called the *weight sequence* of T.

For a given rooted tree T, let H be the ordered tree corresponding to T that has the heaviest depth sequence $L(H)$ among all the ordered trees corresponding to T. Then H is called the *left-heavy embedding* of T, and $L(H)$ is the *left-heavy depth sequence* of T. Left-heavy embedding is uniquely defined for any rooted tree T, and no two distinct rooted trees share the same left-heavy embedding [19].

Chapter 3

Enumeration on Block-Cutpoint trees

In a connected graph G, a vertex v is called a *cutpoint* if the removal of v leads to the disconnection of G. A *block* is a maximal connected subgraph of G free of cutpoints. For a connected graph G, let $\mathcal{B}(G)$ denotes the set $\{B_1, B_2, \ldots\}$ of blocks, and $\mathcal{C}(G)$ denotes the set $\{c_1, c_2, \ldots\}$ of cutpoints of G. The *block-cutpoint* graph $T(G) = (V(T), E(T))$ of G is characterized by $V(T) = \mathcal{B}(G) \cup \mathcal{C}(G)$ and $E(T) = \{\{c, B\} \mid c \in \mathcal{C}(G), B \in \mathcal{B}(G),\ c \text{ belongs to } B \text{ in } G\}$.

Harary proved the following theorem in [13]:

Theorem 1. *A graph T is a block-cutpoint graph of a graph G if and only if T is a tree in which the distance between any two leaves is even.*

We call the block-cutpoint graph $T(G)$ the *block-cutpoint tree* (BC-tree in short) of G in the following context. It is easy to observe that any leaf of a BC-tree is a block vertex, namely a vertex in $\mathcal{B}$.

Let T denote a tree. The *eccentricity* of a vertex v in a tree T is defined as the maximum distance from vertex v to any other vertex in T. The maximum eccentricity is the *diameter* of T.

A vertex v in T is designated as a *center* of T if it contains the minimum eccentricity. It is widely recognized that any tree possesses either a single center or two adjacent centers (see, for instance, [25]). According to Theorem 1, we can directly obtain the ensuing lemma:

Lemma 1. *Any BC-tree has one center.*

From now on, we consider a BC-tree a rooted tree at its center.

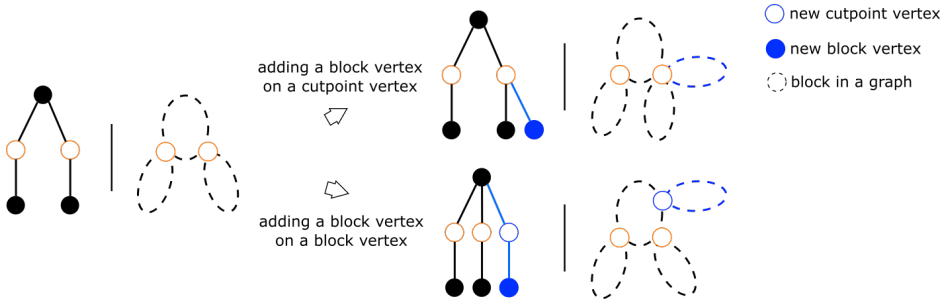

Figure 3.1: Two operations for the generation of BC-trees. The right side of each BC-tree shows the corresponding graph structure.

3.1 Generation Rules for BC-trees

Let T denote a BC-tree with diameter $2k$, and $P = (v_0, v_1, \ldots, v_{2k})$ represent one of the longest paths in T, which has a length of $2k$. Then vertex v_k serves as the center of T.

In a rooted tree T, a subtree rooted at v is denoted by $T(v)$. In such a way, T can be denoted as $T(v_k)$.

We are now able to propose the generation rules for BC-trees (go by Figure 3.1 for illustration):

(1) For a BC-tree T, construct a tree T' from T by adding a new block vertex b to a cutpoint vertex in T.

(2) For a BC-tree T, construct a tree T' from T by adding a new cutpoint vertex c to a block vertex b in T, with a new block vertex b' to c.

The following theorem relates to the generation rules above.

Theorem 2. *A tree T is a BC-tree if and only if T contains a block vertex or it is obtained by a sequence of the rules (1) and (2) from a tree contains a block vertex.*

Proof. We demonstrate the claim by applying induction to the number n of block vertices. When T is a tree that contains a block vertex, T could be either K_1 with one block vertex as its root or K_2 with one block vertex as a leaf and rooted at a cutpoint vertex. Each situation corresponds to a BC-tree. Thus we assume that $n > 1$.

We first show that any tree T generated by a sequence s of the rules is a BC-tree. When the last rule in the sequence s is (1), let b be the block vertex added by the rule. We consider the tree T' obtained from T by removing the

block vertex b from T. Then T' is a tree with $n-1$ block vertices generated by a sequence s' from a block vertex, where s' is the sequence of rules obtained by removing the last rule (1) from s. By the inductive hypothesis, T' is a BC-tree, and b is attached to a cutpoint vertex c in T. Since T' is a BC-tree, any distance between c and b' is odd, where b' is any leaf in T'. Therefore, the distance between b and b' is even, which implies that T is a BC-tree by Theorem 1. On the other hand, when the last rule in s is (2), let b and c be the block and cutpoint vertices added by the rule. We consider the tree T' obtained from T by removing b and c. Then, by a similar argument, we can observe that T' is a BC-tree, and any distance between b and b' for any leaf vertex b' in T' is even. Thus any tree T generated by a sequence s of the rules is a BC-tree.

We next show that any BC-tree T with $n > 1$ vertices can be obtained by a sequence of the rules. Let l be any leaf of T. Then l is a block vertex since T is a BC-tree. Since $n > 1$, l has a neighbor c, which is a cutpoint vertex with $\deg(c) > 1$. We consider two cases.

- **Case 1:** $\deg(c) > 2$. Let T' be a tree obtained from T by removing l. Since $\deg(c) > 2$, l holds at least one sibling. Then, removing l from T does not break the properties of BC-tree; namely, T' is a BC-tree with $n-1$ block vertices. Thus, by the inductive hypothesis, it is obtained by a sequence s' of the rules. It is easy to see that we can obtain T from T' by adding l according to rule (1).

- **Case 2:** $\deg(c) = 2$. Let l' be the other neighbor of c. That is, $N(c) = \{l, l'\}$. Let T' be a tree obtained from T by removing l and c. Then in the same argument, T' is a BC-tree with $n-1$ block vertices, which can be obtained by a sequence of rules. We can obtain T from T' by using the rule (2).

□

It is important to observe that in the generation rules of BC-trees, both (1) and (2) lead to the increase of precisely one block vertex. In the rest of this paper, the term "adding a block vertex" denotes the implementation of either generation rule (1) or (2).

3.2 Enumeration on Ordered BC-trees

In this chapter, our aim is to enumerate all ordered BC-trees with at most n block vertices.

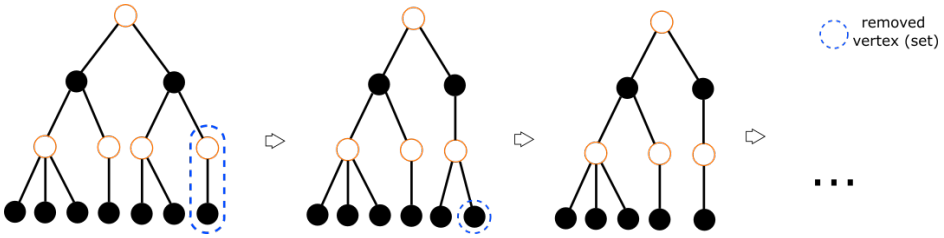

Figure 3.2: Remove Operation on a BC-tree

3.2.1 Family Tree

In [2], Avis and Fukuda introduced the reverse search technique for enumeration problems. In reverse search, the target set's parent-child relationship forms the family tree. Except for the root, each family tree vertex is in the target set and has a unique parent. After traversing the family tree, all target set elements are enumerated to avoid duplicates and omissions.

Nakano proposed an enumeration algorithm for ordered trees that outputs in constant time for each instance in [15]. We here extend this algorithm to enumerate ordered BC-trees, resulting in a constant enumeration algorithm for ordered BC-trees.

Consider an ordered BC-tree T with root r. Let $P = (r = v_0, v_1, v_2, \ldots, v_i)$ represent a path in T. Then P is called the *right path* of T if, for $1 \leq j \leq i$, v_j is the rightmost vertex among its siblings and v_i is a leaf in T.

Observe that in the context of BC-trees, v_i always corresponds to a block vertex. Then, by repeatedly removing such v_i (and the cutpoint vertex adjacent to v_i if necessary), we finally obtain a BC-tree that contains exactly one block vertex; see Figure 3.2.

By Lemma 1, any rooted BC-tree has exactly one root. Note that the root of a BC-tree could be either a block vertex or a cutpoint vertex. Let S_n be the set of ordered BC-trees with at most n vertices; we define $\mathcal{T}_{br}^n$ ($\mathcal{T}_{cr}^n$, respectively) as the family tree of all ordered BC-trees with at most n vertices and rooted at a block vertex (rooted at a cutpoint vertex, respectively). Then we have $S_n = V(\mathcal{T}_{br}^n) \cup V(\mathcal{T}_{cr}^n)$.

Let $T, T' \in \mathcal{T}_{br}^n$ ($\mathcal{T}_{cr}^n$, respectively), we define T' as the parent of T in $\mathcal{T}_{br}^n$ ($\mathcal{T}_{cr}^n$, respectively) if T' can be obtained from T by removing the rightmost block vertex. Let the BC-tree with one block vertex be the root of $\mathcal{T}_{br}^n$ and the BC-tree with a cutpoint vertex as its root and a block vertex be the root of $\mathcal{T}_{cr}^n$. Easy to observe that the element with depth d in either $\mathcal{T}_{br}^n$ or $\mathcal{T}_{cr}^n$ represents an ordered BC-tree with $d+1$ block vertices.

3.2.2 Enumeration Algorithm

Based on the previously discussed definition of family trees, we propose an enumeration algorithm for ordered BC-trees.

Algorithm 1 Enumeration on Ordered BC-trees

Input: An integer n
Output: All Ordered BC-trees with at most n block vertices
1: Initial T_1 as K_1 and T_2 as $P_2 = (v_0, v_1)$ rooted at v_0
2: **Enum-Ordered-BCTree**(T_1)
3: **Enum-Ordered-BCTree**(T_2)

procedure Enum-Ordered-BCTree(T)
1: Output T
2: **if** $|V(T)| < n$ **then**
3: **for each** v in the right path of T **do**
4: add a block vertex on v to obtain T'
5: **Enum-Ordered-BCTree**(T')
6: **end for**
7: **end if**
end procedure

Theorem 3. *Ordered BC-trees with at most n block vertices can be enumerated by Algorithm 1 in constant time for each.*

Proof. Our algorithm enumerates BC-trees rooted at block vertices and cut-point vertices as separate classes. We maintain the depth sequence of T while outputting only the difference from the previous one in Line 1 of the procedure **Enum-Ordered-BCTree**, specifically the depth of the leaf on the right path of the current tree. By recursively generating and outputting BC-trees, all satisfied ordered BC-trees can be output in constant time for each. □

3.3 Enumeration on Unordered BC-trees

3.3.1 Canonical Representation

Section 4 gives Algorithm 1, which enumerates all satisfied ordered BC-trees. Using Algorithm 1 to enumerate unordered BC-trees leads to duplicate outputs. We propose the canonical representation of unordered BC-trees for avoiding duplicate cases.

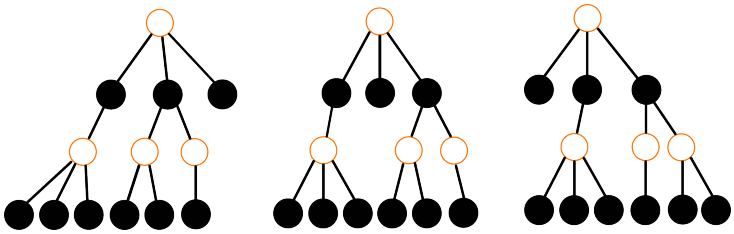

Figure 3.3: Three isomorphic unordered BC-trees. When considered as ordered BC-trees, the left one serves as the canonical representation due to the definition.

In an unordered BC-tree T, the *canonical representation* refers to the BC-tree structure characterized by a left-heavy embedding, as illustrated in Figure 3.3. The set of unordered BC-trees is considered a subset of the set of ordered BC-trees.

3.3.2 Family Tree

The canonical representation allows the definition of the family tree for unordered BC-trees in a manner similar to that of ordered BC-trees.

Let S'_n be the set of unordered BC-trees in canonical representation with at most n vertices, $\mathcal{T}_{br}^{n\prime}$ ($\mathcal{T}_{cr}^{n\prime}$, respectively) as the family tree of all unordered BC-trees in canonical representation with at most n vertices and rooted at a block vertex (rooted at a cutpoint vertex, respectively). Assume $T, T' \in S'_n$, then the children of any vertex in T and T' are assigned an order. Let $(r = v_0, v_1, v_2, \ldots, v_i)$ be the right path of T. We say T' is the parent of T in either $\mathcal{T}_{br}^{n\prime}$ or $\mathcal{T}_{cr}^{n\prime}$ if T' can be obtained from T by removing v_i (and the cutpoint vertex adjacent to v_i if necessary). By repeatedly applying the remove operation, we finally obtain a BC-tree that contains exactly one block vertex.

It has been observed that when enumerating unordered BC-trees using the same technique as for ordered BC-trees, the addition of a block vertex to T' could end up in a BC-tree that is not in canonical representation in particular cases. Consequently, we extend the enumeration algorithm proposed in [17] through the enumeration of unordered BC-trees.

Let $T \in S'_n$, v be a vertex in the right path of T, and v' be the left sibling of v in T, if it exists. We consider all three cases for depth sequences $L(T(v))$ and $L(T(v'))$ as follows:

- Case 1: $L(T(v)) = L(T(v'))$.

In such a case, adding a block vertex on any vertex on the right path of T' breaks the left-heavy manner at vertex v.

- Case 2: $L(T(v))$ is not the prefix of $L(T(v'))$.

  Note that $L(T(v)) < L(T(v'))$ satisfies since T' is in canonical representation. In such a case, adding a block vertex on any vertex on the right path of T' does not break the left-heavy manner on v.

- Case 3: $L(T(v)) \neq L(T(v'))$ but $L(T(v))$ is the prefix of $L(T(v'))$.

  Assume l is the length of $L(T(v))$ and let d be the $(l+1)$th element in $L(T(v'))$. Then it is obvious that the depth of the next vertex should be less equal than d.

Due to the case analysis, we have the following lemma.

Lemma 2. *Let $T \in S'_n$ and $(v_0, v_1, \cdots, v_l)$ be the right path of T, v'_i be the left vertex of v_i if exists. The following propositions are equivalent:*

(1) T' obtained from T by adding a block vertex on v_i is in canonical representation;

(2) $L(T(v_j)) < L(T(v'_j))$ for $2 \leq j \leq i$.

Proof. (1) $\rightarrow$ (2). Let T' is obtained from T by adding a block vertex on v_i in canonical representation. Namely, for $2 \leq j \leq$, $L(T(v_j)) \leq L(T(v'_j))$ satisfies in T' for any j. Obviously, $L(T(v_j)) < L(T(v'_j))$ establishes in T.

(2) $\rightarrow$ (1). Let $2 \leq j \leq i$, since $T \in S'_n$, $L(T(v_j)) \leq L(T(v'_j))$ holds. Assume there exists such j that satisfies $L(T(v_j)) = L(T(v'_j))$, then adding a block vertex on v_j leads to $L(T(v_j)) > L(T(v'_j))$, namely T is not in canonical representation, a contradiction.

□

3.3.3 Enumeration Algorithm

Here, we are able to propose the enumeration of unordered BC-trees.

Algorithm 2 Enumeration on Unordered BC-trees

Input: An integer n

Output: All Unordered BC-trees with at most n block vertices

1: Initial T_1 as K_1 and T_2 as $P_2 = (v_0, v_1)$ rooted at v_0
2: **Enum-Unordered-BCTree**(T_1)
3: **Enum-Unordered-BCTree**(T_2)

procedure Enum-Unordered-BCTree(T)

1: Output T ▷ Assume $(v_0, v_1, \cdots, v_l)$ be the right path of T
2: **if** $|V(T)| < n$ **then**
3: $end \leftarrow l$
4: **for** $i \leftarrow 1$ to end **do**
5: **if** $L(T(v_i)) = L(T(v'_i))$ **then** ▷ Case 1
6: **Break**
7: **end if**
8: Update informations related to v_i
9: **if** $L(T(v_i))$ is the prefix of $L(T(v'_i))$ **then**
10: $end \leftarrow d - 1$ ▷ Case 3
11: **end if**
12: add a block vertex on v_i to obtain T'
13: **Enum-Unordered-BCTree**(T')
14: **end for**
15: **end if**
end procedure

Theorem 4. *Unordered BC-trees with at most n block vertices can be enumerated by Algorithm 2 in constant time for each.*

Proof. The proof simply corresponds to that of Algorithm 1, with an emphasis on the distinct parts. In Case 3 (lines 8-9), we update *end* to prune the process and avoid obtaining BC-trees that are not in canonical representation.

Each BC-tree in canonical representation is derived from the subtree containing one block vertex (K_1 or K_2), corresponding to Case 3. Increasing the block vertex repeatedly maintains Case 3 or transitions to Case 1 or Case 2.

We maintain the depth sequence of the current BC-tree T, and we also maintain the start and end indexes of $L(T(v'))$ for each vertex v with d in Case 3, if applicable. We can then complete lines 5 and 8 in constant time.

Lines 5-6 indicate the break from the current loop in Case 1. In line 7, we update $L(T(v_i))$ and the case related to the current v_i. Additionally, we update $L(T(v'_i))$ and d if v_i is in existence. Subsequently, we can amortize the update of the right path from linear time to constant time for each output. In particular, each output requires constant time, while the algorithm uses $O(n)$ space. □

Here, we would like to point out an important detail regarding line 3 of Algorithm 2. When enumerating BC-trees that are rooted at cutpoint vertices, consider the relationship between a BC-tree and its associated graph, not all of these trees correspond to actual graphs. Specifically, among all

the BC-trees enumerated by line 3, those in which the root vertex v_0, a cutpoint vertex, has only one child node v_1 do not correspond to any graph (other newly added cutpoint vertices must be adjacent to two block vertices). While these tree structures are valid in terms of the BC-tree definition, they should be handled with care in practical applications.

Chapter 4

Enumeration on Weighted Trees

In order to enrich the framework, we focus on the details of the second stage in this chapter. Specifically, in this chapter, we generalize the enumeration problem of weighted BC trees in the second stage into the enumeration problem of weighted trees for discussion, and proposed an enumeration algorithm for weighted trees such that each satisfied weighted tree can be enumerated in amortized constant time for each.

4.1 Reduced Set Trees

4.1.1 Reduced Set Tree

For a given rooted tree $T = (V, E)$ in left-heavy embedding, the *reduced set tree* (*RSTree* for short) T' of T is a tree structure that compacts isomorphic subtrees in T.

The RSTree T' is obtained from a given tree T as follows.

For T, we recursively iterate vertices by their depth. For depth d, a temporary list is constructed to store the vertex-label pairs (v, l) for vertices in V_d ordered from left to right, where l encodes the label of the subtree rooted at v and V_d is the set of vertices with depth d. Leaves are assigned empty labels [], while the labels of internal nodes are formed from the labels of their children in order. These labels are essentially representations of subtree structures, constructed recursively from the leaves up. Next, we map each distinct label to a unique integer identifier to compress the representation for each label. Such compressed labels allow us to check whether two sibling subtrees are isomorphic in linear time. Note that the steps above do not

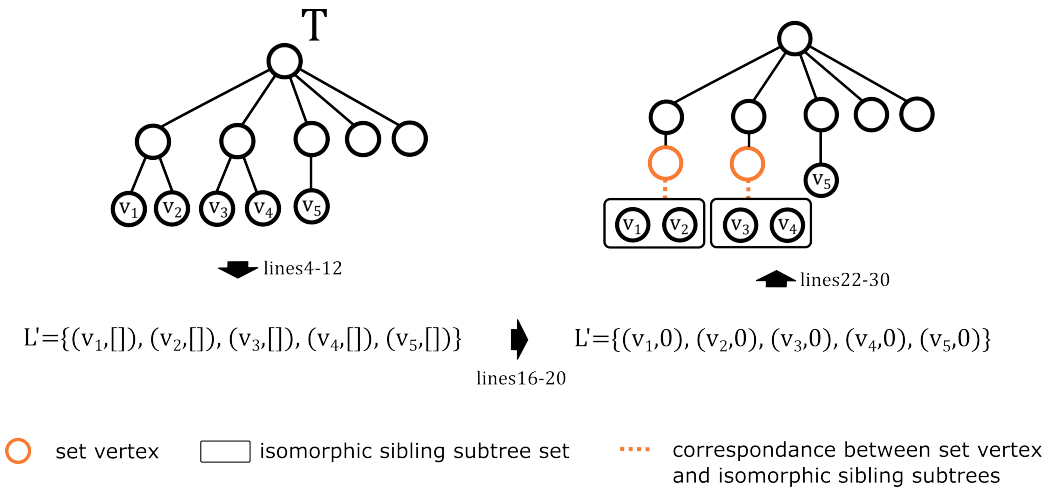

Figure 4.1: An illustration for Algorithm 3 when $d = height(T)$ for a given rooted tree T.

change the order for vertices in T. At last, we compare the compressed labels for pairs of siblings to check the isomorphism of the corresponding sibling subtrees. Once multiple sibling subtrees are isomorphic, we remove all these subtrees and store them into a newly obtained vertex called the *set vertex*. For a set vertex sv, the corresponding subtrees that are removed are denoted by a subtree set $S(sv) = \{T(v_1), T(v_2), \ldots, T(v_m)\}$. Due to all subtrees in such a set being isomorphic, it is not necessary to keep their order. By recursively applying such operations from bottom to top, we can obtain the RSTree T' from T.

Our idea above is based on the AHU algorithm (see Example 3.2, [1]), which is able to determining the isomorphism of two rooted trees in linear time. Hereafter, for a given rooted tree T in left-heavy embedding, we give Algorithm 3 to obtain an RSTree T' from T. Figure 4.1 gives an illustration for the process when $d = height(T)$ for a given rooted tree T.

Thus we have Theorem 5.

Theorem 5. *For a given ordered tree T, Algorithm 3 generates the RSTree T' of T in $O(n)$ time.*

Proof. In lines 3-12, all vertices with depth d are traversed only once to construct L', a temporary list of vertex-label pairs for vertices in V_d. In line 13, the time cost for sorting L in lexicographical order is proportional to the number of tuples in L.

Then in lines 16-20, each label constructed above is compacted into integers by their order to construct a new vertex-label list.

Algorithm 3 Generation of the reduced set tree of a given tree

Input: Tree T.

Output: The reduced set tree T' of T.

$L \leftarrow$ empty list
for $d \leftarrow height(T)$ to 0 **do**
    $L' \leftarrow$ empty list
    for pair (v, k) in L **do**
        $p \leftarrow$ parent of v
        if p is a vertex in L' **then**
            Append k to the label of p in L'
        else
            Append new pair $(p, [k])$ to L'
        end if
    end for
    for v in V_d **do**
        if v is a leaf **then**
            Append pair $(v, [])$ to L'
        end if
    end for
    $L \leftarrow L'$ and sort L in lexicographical order
    $L' \leftarrow$ empty list
    $idx \leftarrow 0$, $label \leftarrow$ label of first pair in L
    for pair (v, l) in L in order **do**
        if $l \neq label$ **then**
            $idx \leftarrow idx + 1$
            $label \leftarrow l$
        end if
        Append pair (v, idx) to L'
    end for
    $L \leftarrow L'$
    for pair (v, l) in L in order **do**
        if v^- exists and $l = l^-$ **then**
            if v^+ not exists or $l \neq l^+$ **then**
                add set vertex v_{set} as the right sibling of v
                $v_{cur} \leftarrow v$, $l_{cur} \leftarrow l$
                while v_{cur}^- exists and $l_{cur} = l_{cur}^-$ **do**
                    remove subtree $T(v_{cur})$ and add $T(v_{cur})$ to $S(v_{set})$
                    $v_{cur} \leftarrow v^-$, $l_{cur} \leftarrow l^-$
                end while
                remove subtree $T(v_{cur})$ and add $T(v_{cur})$ to $S(v_{set})$
            end if
        end if
    end for
end for

Lastly, in lines 22-30, we traverse pairs in L to check if isomorphic subtrees exist. If so, we remove isomorphic subtrees and store them at a newly added set vertex.

The correctness of our algorithm can be referenced by the AHU algorithm. The time complexity for both assigning labels and integers to all vertices is proportional to the number of vertices in T. Also, each subtree rooted at a vertex in T is possibly removed at most once by lines 28-30. Namely, the operation to each vertex is restricted within constant time, and for a given ordered tree T, Algorithm 3 can be done in $O(n)$ time. □

It is easy to see that the RSTrees T_1' and T_2' are with the same structure if and only if T_1 and T_2 are isomorphic. We here note that two subtrees T_1' and T_2' with set vertices are defined to be *isomorphic* only when the original corresponding subtrees of T_1 and T_2 are isomorphic.

Lemma 3. *For a given rooted tree T in left-heavy embedding and an arbitrary weight sequence suited to T, there is precisely one corresponding weighted tree if and only if the RSTree T' is isomorphic to T.*

Proof. We first consider the if part. Then, for T and weight sequence W, there is precisely one corresponding weighted tree. Thus, there are no two isomorphic sibling subtrees in T. Namely, T is isomorphic to the RSTree of T.

The only-if part is proved by contradiction. Assume T is not isomorphic to the RSTree of T, and for a given arbitrary weight sequence $W(T)$ suited to T, there is precisely one corresponding weighted tree.

Let $(v_1, \ldots, v_i, v_{i+1}, \ldots, v_{i+k}, \ldots, v_j, v_{j+1}, \ldots, v_{j+k}, \ldots, v_m)$ be the preorder of T, there must exist at least one pair of siblings v_i and v_j such that subtrees induced by $(v_i, v_{i+1}, \ldots, v_{i+k})$ and $(v_j, v_{j+1}, \ldots, v_{j+k})$ are isomorphic for some k. Let $W(T) = (w_1, \ldots, w_i, w_{i+1}, \ldots, w_{i+k}, \ldots, w_j, w_{j+1}, \ldots, w_{j+k}, \ldots, w_m)$, consider l that satisfies $w_l \neq w_{(j-i)+l}$ for $i \leq l \leq i+k$, namely by changing the order of subtrees induced by $(v_i, v_{i+1}, \ldots, v_{i+k})$ and $(v_j, v_{j+1}, \ldots, v_{j+k})$, we can obtain $W'(T) = (w_1, \ldots, w_j, w_{j+1}, \ldots, w_{j+k}, \ldots, w_i, w_{i+1}, \ldots, w_{i+k}, \ldots, w_m)$, another weight sequence of T. It is obvious that $W(T) \neq W'(T)$ since there exists such l which satisfies $w_l \neq w_{(j-i)+l}$ for $i \leq l \leq i+k$, which gives a contradiction. □

For an RSTree T' isomorphic to its original tree T, T is called the *sibling-distinct tree*.

4.2 Enumeration on weighted sibling-distinct trees

Here, we first deal with the enumeration algorithm of weighted sibling-distinct trees. The enumeration of the weighted sibling-distinct trees is based on the reverse search. We denoted by $\mathcal{T}_w(T) = (V_w(T), E_w(T))$ the *family tree* for all possible weight sequences of a given sibling-distinct tree T with weight w.

We first give the definition of binary relations $\overset{w}{=}$, $\overset{w}{>}$ and $\overset{w}{<}$ on pairs of weighted sibling-distinct trees. Note that for weighted single vertices v_1 and v_2, we simply define $v_1 \overset{w}{>} v_2$ ($v_1 \overset{w}{=} v_2$ and $v_1 \overset{w}{<} v_2$, respectively) if $w(v_1) > w(v_2)$ ($w(v_1) = w(v_2)$ and $w(v_1) < w(v_2)$, respectively).

Let T_1 and T_2 be two weighted isomorphic sibling-distinct trees with the same weight w, namely, $W(T_1)$ and $W(T_2)$ are in the same family tree $\mathcal{T}_w(T)$. We say $T_1 \overset{w}{=} T_2$ if $W(T_1) = W(T_2)$. $T_1 \overset{w}{>} T_2$ if $W(T_1)$ appears before $W(T_2)$ in the pre-order traversal on $\mathcal{T}_w(T)$. $T_1 \overset{w}{<} T_2$ is defined in the same way.

Let $W \in V_w(T)$ with $W = (w_1, w_2, \ldots, w_p, \ldots, w_n)$ and p be the maximal index that satisfies $w_p > 0$, that is, $w_{p+1} = \cdots = w_n = 0$. Consider the weight sequence $(w_1, w_2, \ldots, w_{p-1} + 1, w_p - 1, \ldots, w_n)$, which is obtained from W. Then we have the following lemma.

Lemma 4. *Let $W \in V_w(T)$ with $W = (w_1, w_2, \ldots, w_p, \ldots, w_n)$, then the weight sequence $(w_1, w_2, \ldots, w_{p-1} + 1, w_p - 1, \ldots, w_n)$ which is derived from W is also in $V_w(T)$.*

Proof. The obtainment does not change the total weight and the length of the sequence. Thus, the newly obtained weight sequence is also in $V_w(T)$. □

In the context of the family tree of the weighted tree, the weight sequence $(w_1, w_2, \ldots, w_{p-1} + 1, w_p - 1, \ldots, w_n)$ is defined to be the *parent sequence* of W in $\mathcal{T}_w(T)$, denoted by $P(W)$. W is called the *child sequence* of $P(W)$ in $\mathcal{T}_w(T)$.

For any weight sequence $W \in \mathcal{T}_w(T)$, by repeatedly generating $P(W)$, we can obtain a unique sequence $W, P(W), P(P(W)) \ldots$ of weight sequences in the family tree $\mathcal{T}_w(T)$. All these sequences eventually ended at the weight sequence $(w, 0, 0, \ldots, 0)$. By merging all these sequences, we can construct $\mathcal{T}_w(T)$. The enumeration of all weight sequences with the given T and weight w can be applied by traversing $\mathcal{T}_w(T)$ and outputting all elements in $\mathcal{T}_w(T)$. Note that once T is restricted to a sibling-distinct tree, it is not necessary to consider $L(T)$ when enumerating the weight sequence of T. Figure 4.2 shows

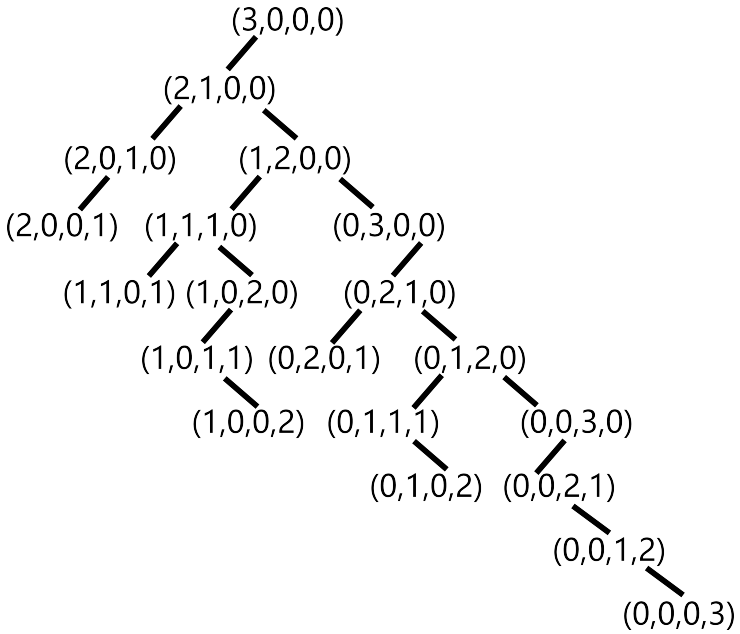

Figure 4.2: An illustration of the family tree for all weight sequences of the weighted sibling-distinct tree with total weight 3 and 4 vertices.

an illustration of the family tree for all weight sequences of the weighted sibling-distinct tree with total weight 3 and 4 vertices.

Hereafter, we consider the parent-child relationship in the family tree, namely the generation from the parent sequence to the child sequence. We have the following lemma:

Lemma 5. *Let $W, W' \in \mathcal{T}_w(T)$ with $W = (w_1, w_2, \ldots, w_p, \ldots, w_n)$, where p is the maximal index that satisfies $w_p > 0$ for $1 \leq p \leq n$. Then W' is the child sequence of W if and only if either (1) $W' = (w_1, w_2, \ldots, w_{p-1} - 1, w_p + 1, \ldots, w_n)$ when $p > 1$ and $w_{p-1} > 0$, or (2) $W' = (w_1, w_2, \ldots, w_p - 1, 1, \ldots, w_n)$ when $p < n$.*

Proof. It is easy to observe that the parent sequence of $(w_1, w_2, \ldots, w_{p-1} - 1, w_p+1, \ldots, w_n)$ and $(w_1, w_2, \ldots, w_p-1, 1, \ldots, w_n)$ is $(w_1, w_2, \ldots, w_p, \ldots, w_n)$ based on the definition of the parent-child relation in the family tree, which proves the necessity of the lemma. For sufficiency, assume W' is the child of W with $W' = (w'_1, \ldots, w'_p, \ldots, w'_n)$, then we have $W = (w'_1, \ldots, w'_{p-1} + 1, w'_p - 1, \ldots, w'_n)$ due to the definition of the parent-child relationship. We consider the weight $w'_p - 1$ in W in two cases: (1)$w'_p - 1 > 0$. In such a case, W and W' hold the same p. Due to W and W', $w'_{p-1} > 0$ and $p - 1 \geq 1$ hold, namely $w'_{p-1} + 1 > 1$ and $p > 1$, which corresponds to (1) in Lemma 5;

(2)$w'_p - 1 = 0$. In such a case, p is increased by 1 from W to W'. Therefore, $p < n$ holds in W, which corresponds to (2) in Lemma 5. □

Here, we propose Algorithm 4 to enumerate all possible weight sequences for suit T and weight w in constant time. Note that each output corresponds to a pair of index and weight of a vertex.

Algorithm 4 Enumeration on weighted sibling-distinct trees

Input: T: a sibling-distinct tree with order n; w: the total weight T.
Output: all weighted trees suit to T of weight w.

1: Initial $W(T) = (w, 0, 0, \ldots, 0)$ ▷ $|W| = n$
2: **Enum-SDTree**($W(T)$, 1)

procedure Enum-SDTree(W, p) ▷ $W = (w_1, w_2, \ldots, w_p, \ldots, w_n)$ is the current weight sequence, p is the maximal index that satisfies $w_p > 0$
1: **if** $p = 1$ **then**
2: Output pair (1, w_1)
3: **else**
4: Output pairs ($p - 1$, w_{p-1}) and (p, w_p)
5: **end if**
6: **if** $p > 1$ and $w_{p-1} > 0$ **then**
7: **Enum-SDTree**($(w_1, w_2, \ldots, w_{p-1} - 1, w_p + 1, \ldots, w_n)$, p)
8: **end if**
9: **if** $p < n$ **then**
10: **Enum-SDTree**($(w_1, w_2, \ldots, w_p - 1, 1, \ldots, w_n)$, $p + 1$)
11: **end if**
end procedure

Theorem 6. *For a given sibling-distinct tree T and weight w, Algorithm 4 enumerates all weighted trees that suit T and weight w in constant time per tree.*

Proof. Algorithm 4 initializes the weight sequence as the root of the family tree and calls the procedure **Enum-SDTree**(). In procedure **Enum-SDTree**(), time complexity follows the cost of applying DFS on the family tree. In lines 1-5, it only outputs the difference from the previous one, which outputs at most two vertex weights and the corresponding vertex index for each output. Based on Lemma 5, lines 6-11 recursively call procedure **Enum-SDTree**() to traverse the whole family tree. It is simple to observe that each recursion runs in only constant steps. Thus, each weight sequence can be enumerated in constant time. □

4.3 Enumeration on Weighted RSTrees

The fundamental strategy of the RSTree enumeration, which relies on the concept of mutual recursion, is illustrated in this section.

4.3.1 Enumeration on Weighted Set Vertices

For a given weighted set vertex sv of weight w, the enumeration of a weighted set vertex can be separated into two parts. We first enumerate the weight sequence of sv, then enumerate the weighted RSTrees of $S(sv)$ in canonical order.

Weight Sequence of Set Vertex

Firstly, we consider the enumeration of the weight sequences of a weighted set vertex. For the set vertex sv of weight w, we aim to output all weight sequences satisfying sv with sum w, where the length of the weight sequence corresponds to the number of subtrees in sv. The enumeration of the weighted set vertex uses a similar framework as the enumeration of the weighted tree.

Let sv be a set vertex that corresponds to the subtree set $S(sv) = \{T(v_1), T(v_2), \ldots, T(v_m)\}$. Even subtrees $T(v_1), T(v_2), \ldots, T(v_m)$ are isomorphic, the weight sequence consists of the weight sequences of such subtrees with different orders that are numerous. In such a case, we define the *canonical order* of isomorphic subtrees as $T(v_i) \overset{w}{\geq} T(v_{i+1})$ for $1 \leq i \leq m-1$, and the *weight sequence* of the set vertex is denoted as $W(S(sv)) = (w(T(v_1)), w(T(v_2)), \ldots, w(T(v_m)))$.

We define the family tree $\mathcal{T}_w(sv) = (V_w(sv), E_w(sv))$ such that all possible weight sequences suiting $S(sv)$ of weight w are elements in $\mathcal{T}_w(sv)$. For weighted set vertices sv_1 and sv_2, the weight sequence of sv_1 is denoted as $W(S(sv_1)) = (w(T(v_1)), w(T(v_2)), \ldots, w(T(v_m)))$ and the weight sequence of sv_1 is denoted as $W(S(sv_1)) = (w(T(v_1)), w(T(v_2)), \ldots, w(T(v_m)))$ with $W(S(sv_1)), W(S(sv_2)) \in V_m(sv)$. We say $sv_1 \overset{w}{=} sv_2$ if $T(v_i) \overset{w}{=} T(v_i')$ for $1 \leq i \leq m$, and $sv_1 \overset{w}{>} sv_2$ if (1)$W(S(sv_1))$ shows before $W(S(sv_2))$ in the pre-order traversal on $\mathcal{T}_w(sv)$, or (2)$W(S(sv_1)) = W(S(sv_2))$, $T(v_i) \overset{w}{=} T(v_i')$ and $T(v_{i+1}) \overset{w}{>} T(v_{i+1}')$ for some $1 \leq i < m$. $sv_1 \overset{w}{<} sv_2$ is defined in the same way.

Now we focus on the parent-child relationship in $\mathcal{T}_w(sv)$.

Let $WS = (w_1, w_2, \ldots, w_p, \ldots, w_m)$ where p is the maximal index that satisfies $w_p > 0$. Recall that $w_i \geq w_{i+1}$ holds for $1 \leq i \leq m-1$. Consider

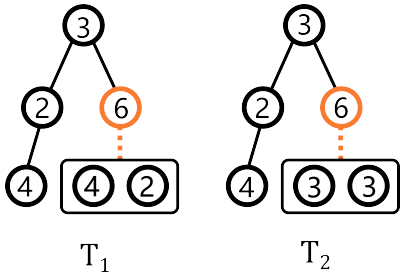

Figure 4.3: An illustration showing two weighted RSTrees, T_1 and T_2, where $T_1 \overset{w}{>} T_2$ holds. Despite T_1 and T_2 containing identical weight sequences, the disparity in the weight sequence of sv results in $T_1 \overset{w}{>} T_2$, as defined in Case 3, Section 3.2.

the weight sequence $(w_1 + 1, w_2, \ldots, w_p - 1, \ldots, w_m)$, which is derived from WS. We have the following lemma:

Lemma 6. *Let $WS \in V_w(sv)$ with $WS = (w_1, w_2, \ldots, w_p, \ldots, w_m)$, then the weight sequence $(w_1 + 1, w_2, \ldots, w_p - 1, \ldots, w_m)$ which is derived from WS is also in $V_w(sv)$.*

Proof. The operation does not change the total weight or the length of the sequence. Thus, the newly obtained weight sequence is also in $V_w(sv)$. □

The weight sequence $(w_1 + 1, w_2, \ldots, w_p - 1, \ldots, w_m)$ is called the *parent sequence* of WS in the context of the weighted set vertex. It is indicated by $P(WS)$ in $\mathcal{T}_w(sv)$. For every weight sequence $WS \in V_w(sv)$, we can generate a unique sequence $WS, P(WS), \ldots$ in the family tree $\mathcal{T}_w(sv)$ by repeatedly creating $P(WS)$. Furthermore, all of these sequences eventually arrive at the endpoint $(w(sv), 0, \ldots, 0)$. Similarly, we can generate the family tree $\mathcal{T}_w(sv)$ by combining all of these sequences. The weight sequence $(w(sv), 0, \ldots, 0)$ is considered to be the root of $\mathcal{T}_w(sv)$. Figure 4.4 shows an illustration.

Pay attention that we implement a distinct parent-child relationship compared to the case of the weighted sibling-distinct tree. In the weighted sibling-distinct tree, the weights assigned to vertices are independent, while in the weighted set vertex, the weight sequence has to retain canonical order. Utilizing the same parent-child relationship results in instances that contradict the canonical order, specifically a series of duplicated outputs. Consequently, we use a different definition on the family tree of the weighted set vertices to ensure that the weight sequence is consistently ordered in canonical order.

Consider generating the weight sequence of the sibling subtree sequence from its parent sequence; we have Lemma 7.

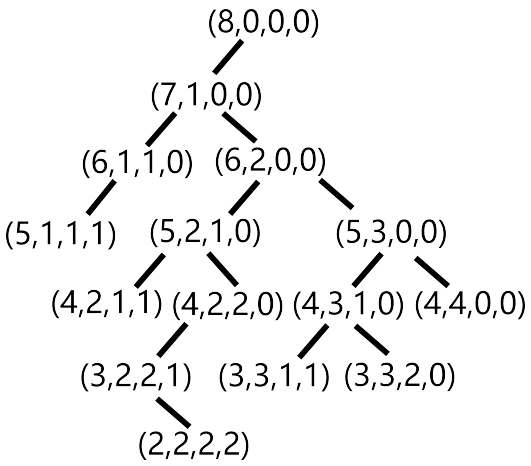

Figure 4.4: The family tree for all weight sequences in the set has four isomorphic subtrees, and the total weight of the corresponding set vertex is 8.

Lemma 7. *Let $WS, WS' \in V_w(sv)$ with $WS = (w_1, w_2, \ldots, w_p, \ldots, w_m)$, where p is the maximal index that satisfies $w_p > 0$. Then WS is the parent sequence of WS' in $\mathcal{T}_w(sv)$ if and only if either (1) $WS' = (w_1 - 1, w_2, \ldots, w_p + 1, \ldots, w_m)$ when $p > 2$, $w_p < w_{p-1}$ and $w_1 > w_2$, (2) $WS' = (w_1 - 1, w_2, \ldots, w_p, 1, \ldots, w_m)$ when $1 < p < s$ and $w_1 > w_2$, or (3) $WS' = (w_1 - 1, w_2 + 1, \ldots, w_m)$ when $1 \leq p \leq 2$ and $w_1 > w_2 + 1$.*

Proof. The necessity is obvious. We demonstrate sufficiency. Assume WS' is the child of WS with $WS' = (w'_1, \ldots, w'_p, \ldots, w'_m)$, then we have $WS = (w'_1 + 1, \ldots, w'_p - 1, \ldots, w'_m)$ due to the above definition of the parent-child relationship. In the following cases, we consider the weight $w'_p - 1$ in WS: (1) $p > 2$ and $w'_p - 1 > 0$. Then we have $w'_1 - 1 \geq w'_2$ and $w'_{p-1} \geq w'_p - 1$. Therefore, $w'_1 > w'_2$ and $w'_{p-1} > w'_p$ hold. This corresponds to (1) in Lemma 7. (2) $p > 2$ and $w'_p - 1 = 0$. In this case, p is increased by 1 from WS to WS', indicating that p of WS can be at least 2. This case corresponds to (2) in Lemma 7. (3) $p = 2$. Specifically, $w'_1 + 1 \geq w'_2 - 1$ is held by WS'. Note that in this case, p of WS can be either 1 or 2. Therefore, we have $w'_1 > w'_2 + 1$ in WS when $1 \leq p \leq 2$, which corresponds to (3) in Lemma 7. The three cases encompass all situations, demonstrating their sufficiency. □

By Lemma 7, we can enumerate all possible weight sequences for a set vertex sv of weight w, and $S(sv)$ contains m subtrees.

Weight Sequences of Isomorphic Subtrees

First, we need to extend the notions of $\overset{w}{=}$, $\overset{w}{>}$ and $\overset{w}{<}$ to the case of weighted RStrees. Let T_1 and T_2 be two weighted isomorphic RSTrees of the same weight, also $W(T_1), W(T_2) \in \mathcal{T}'_w(T)$ holds. Assume $(v_1, v_2, \ldots, v_n)$ and $(v'_1, v'_2, \ldots, v'_n)$ are the pre-orders of T_1 and T_2, respectively. We say $T_1 \overset{w}{=} T_2$ if $v_i \overset{w}{=} v'_i$ for $1 \leq i \leq n$. Note that v_i and v'_i can be either a single vertex or a set vertex. $T_1 \overset{w}{>} T_2$ if (1)$W(T_1)$ appears before $W(T_2)$ in the pre-order traversal on $\mathcal{T}'_w(T)$, or (2)$W(T_1) = W(T_2)$, $v_j \overset{w}{=} v'_j$ and $v_{j+1} \overset{w}{>} v'_{j+1}$ for some $1 \leq j < n$. $T_1 \overset{w}{<} T_2$ is defined in the same way.

Consider a set vertex sv of weight w. Let $S(sv) = \{T_1, T_2, \ldots, T_n\}$ represent the set of isomorphic subtrees associated with sv. Next, in this part, we aim to figure out the approach for enumerating the subtrees in $S(sv)$. In particular, given a tree T and its RSTree T', where $sv \in T'$ contains vertices from v_a to v_b in T, the aim is to enumerate all possible $(w(v_a), w(v_{a+1}), \ldots, w(v_b))$ combinations with a total weight of w. For any two isomorphic subtrees T_i and T_{i+1} in $S(sv)$, where $1 \leq i < n$, since $T_i \overset{w}{\geq} T_{i+1}$ holds, we analyze the relationship between T_i and T_{i+1} in two situations:

Case 1: $w(T_i) > w(T_{i+1})$. In this situation, T_i and T_{i+1} can be individually enumerated with their respective weights.

Case 2: $w(T_i) = w(T_{i+1})$. To ensure that $T_i \overset{w}{>} T_{i+1}$ in this situation, we must handle the weight sequence of T_i with care. It is important to observe that since T_i is isomorphic to T_{i+1} and $w(T_i) = w(T_{i+1})$, the family trees obtained while enumerating T_i and T_{i+1} are the same. In other words, by storing the results when enumerating weighted tree T_i, all satisfied weight sequences of T_{i+1} can be enumerated efficiently.

4.3.2 Enumeration on Weighted Reduced Set Trees

In Section 5.2, we investigated a case of enumerating sibling-distinct trees. In this context, we easily extend the application to include the method of enumeration of the weighted RSTree with set vertices. Note that there is no such lexicographical order that exists among vertices in an RSTree, all set vertices are able to be enumerated separately.

4.3.3 Enumeration Algorithms

Hereafter, we propose the enumeration algorithm of the weighted RSTrees.

Algorithm 5 Enumeration on weighted reduced-set trees

Input: A reduced-set tree T with order n; w the total weight of T.
Output: all weighted trees suit T of weight w.

Initial $W(T) = (w, 0, 0, \ldots, 0)$
Enum-RSTree(T, 1)

---

procedure Enum-RSTree(T, p) ▷ Let $W(T) = (w(v_1), w(v_2), \ldots, w(v_n))$ be the weight sequence of tree T
if $p = 1$ **then**
    Output $w(v_1)$ with index
else
    for $i \to p - 1$ to p **do**
        if v_i is a set vertex **then**
            Enum-SV($(w(v_i), 0, \ldots, 0), 1$)
        else
            Output $w(v_i)$ with index
        end if
    end for
end if
if $p > 1$ and $w_{p-1} > 0$ **then**
    Enum-RSTree($(w_1, w_2, \ldots, w_{p-1} - 1, w_p + 1, \ldots, w_n)$, p)
end if
if $p < n$ **then**
    Enum-RSTree($(w_1, w_2, \ldots, w_p - 1, 1, \ldots, w_n)$, $p + 1$)
end if
end procedure

procedure Enum-SV(W, p) ▷ $W = (w(T_1), w(T_2), \ldots, w(T_p), \ldots, w(T_m))$ is the weight sequence of the current set vertex, and w be the sum of $w(T_i)$ for all $1 \leq i \leq m$.
if $w = 0$ **then**
    return
end if
Enum-RSTree($(w(T_1), 0, \ldots, 0)$, 1)
if $w(T_1) = w(T_2)$ **then**
    for $i \to 2$ to p **do**
        if $w(T_i) = w(T_1)$ **then**
            Enumerate all $W(T_i)$ satisfies $T_i \overset{w}{<} T_{i-1}$
        end if
    end for

11: **end if**
12: **if** $w(T_1) \neq w(T_p)$ **then**
13: **if** $w(T_{p-1}) = w(T_p)$ **then**
14: Enumerate all $W(T_p)$ satisfies $T_{p-1} \overset{w}{<} T_p$
15: **else**
16: **Enum-RSTree**$((w(T_p), 0, \ldots, 0),\ 1)$
17: **end if**
18: **end if**
19: **if** $p \leq 2$ and $w_1 > w_2 + 1$ **then**
20: **Enum-SV**$((w(T_1) - 1, w(T_2) - 1 + 1, \ldots, w(T_m)),\ 2)$
21: **end if**
22: **if** $p > 2$ and $w_p < w_p - 1$ and $w_1 > w_2$ **then**
23: **Enum-SV**$((w(T_1) - 1, w(T_2), \ldots, w(T_p) + 1, \ldots, w(T_m)),\ p)$
24: **end if**
25: **if** $1 < p < m$ and $w_1 > w_2$ **then**
26: **Enum-SV**$((w(T_1) - 1, w(T_2), \ldots, w(T_{p+1}) + 1, \ldots, w(T_m)),\ p + 1)$
27: **end if**
end procedure

Our algorithm consists of two procedures. Procedure **Enum-RSTree** corresponds to the analysis in Section 5.3.2, which enumerates all possible weight sequences of T. Note that in line 4, we only recursively enumerate and output v_{p-1} and v_p; all results of v_j for $1 \leq j \leq p-2$ have already been enumerated and stored to avoid repeatedly enumerating the same subproblems. The procedure **Enum-RSTree** degrades into procedure **Enum-SDTree** when T is sibling-distinct.

Procedure **Enum-SV** corresponds to the analysis in Section 5.3.1, which enumerates all possible weight sequences of sv and recursively enumerates each weighted RSTree in sv. Lines 7 and 10 correspond to Case 2 in the weight sequences of isomorphic subtrees in Section 5.3.1. Note that we only recursively enumerate T_1 and T_p, two subtrees whose weights are changed. Lines 5-7 correspond to the case when $w(T_1) = w(T_2) = \cdots = W(T_j)$ for some $2 \leq j \leq p$. In such a case, the results of $T_2, \ldots, T_j$ need to be re-outputted to satisfy the canonical order.

Thus, we put forward the following theorem:

Theorem 7. *For a given RSTree T and weight w, Algorithm 5 enumerates each weighted tree that suits T of weight w in constant amortized time.*

Proof. Observe that each new result is enumerated only in the procedure **Enum-RSTree** in constant time, as demonstrated in Theorem 6. We then concentrate on the procedure **Enum-SV**. Line 7 executes just when $w(T_i) =$

$w(T_1)$, indicating that all satisfied $W(T_i)$ have been enumerated throughout the enumeration of $W(T_1)$. Consequently, each output costs a constant time delay. In a worst-case scenario where $w(T_1) = w(T_2) = \cdots = w(T_p)$, it requires $O(n)$ time to output the specific weight sequence $W(T_1), W(T_2), \ldots, W(T_p)$, while the output is able to be amortized to $O(1)$ time by following outputs. Line 10 can be executed similarly. Lines 13–18 originate from Lemma 7. □

Note that in Algorithm 3, all satisfied RSTrees can be enumerated in linear time for each, and converting each result over to a tree can be done as well in linear time. By combining Algorithm 3 and Algorithm 5 with the linear time enumeration of ordered trees, we reach the following corollary:

Corollary 1. *For a given pair of positive integers n and w, we can enumerate all weighted trees of vertices less than or equal to n of total weight w in constant amortized time for each.*

Chapter 5

Enumeration Framework by BC-tree

While Chapter 3 concentrated on the enumeration of BC-trees—both ordered and unordered—our objective exceeds simply the enumeration of BC-trees. In numerous graph classes, BC-trees can serve as structural foundations for the graph through cutpoints and blocks. To utilize this structure for the enumeration of wider graph classes, we present an extensive framework in this chapter. This framework extends BC-tree enumeration by systematically constructing graphs from their BC-tree skeletons, allowing the efficient enumeration of classes such as block graphs and cactus graphs according to restraints.

5.1 Enumeration Framework

Consider the enumeration problem for a graph class $\mathcal{G}$; specifically, we aim to enumerate all graphs $G \in \mathcal{G}$ containing no more than n vertices. Our framework contains two phases: the first phase enumerates all BC-trees containing no more than n' block vertices, and the second phase enumerates every possible graph G resulting from each BC-tree enumerated as the final output. The results we obtained in Chapter 3 match seamlessly with the first phase of the framework; we will now concentrate on the second phase.

Assume $T = \{V, E\}$ is a vertex-weighted BC-tree that is derived from $G \in \mathcal{G}$, where a vertex-weighted BC-tree is a BC-tree such that each vertex in the tree is assigned a number. For block vertex $bv \in V$, $w(bv)$, the weight of bv is assigned as the number of vertices in the corresponding block in G, and for cutpoint vertex $cv \in V$, $w(cv)$ is always 1 since it always corresponds to a unique cutpoint in G.

Directly, we have the following lemma:

Lemma 8. *For a given weighted BC-tree $T = \{V, E\}$ derived from a graph G, we have $w = \sum_{v \in V} w(v) - |E|$, where w is the number of vertices in G.*

Proof. Let BV be the set of block vertices and CV be the set of cutpoint vertices in V; we have $V = BV \cup CV$. The total weight of block vertices (cutpoint vertices, respectively) is $\sum_{bv \in BV} w(bv)$ ($\sum_{cv \in CV} w(cv)$, respectively).

In G, every cutpoint vertex belongs to multiple blocks. Correspondingly, in T, cv had been counted $deg(cv)$ times by the adjacent block vertices in $\sum_{bv \in BV} w(bv)$, and every edge adjacent to a cutpoint vertex and a block vertex in T. Therefore, we have $w = \sum_{bv \in BV} w(bv) + \sum_{cv \in CV} w(cv) - |E| = \sum_{v \in V} w(v) - |E|$.

□

The previous lemma establishes a connection between the enumeration problem of graph classes and that of weighted BC-trees. In the second phase of the proposed framework, the most basic information required is the number of vertices in each block.

This framework is extremely useful for enumeration problems in specific graph classes, particularly ones that emphasize the relationships between blocks rather than the internal structure of the blocks. Below, we provide some graph class examples:

5.1.1 Enumeration on Block Graphs

We first focus on the class of block graphs. A characterization of the block graph is first proposed by Harary in [12]. A block graph is defined as a graph in which its blocks are cliques. Namely, a block graph is a connected graph consisting of several cliques organized by cutpoints. The BC-tree distinctly shows the structure of the block graph. As a result, we initially present an approach for enumerating the class of block graphs by the use of BC-trees.

Let T be a BC-tree with n block vertices, with a specified number of vertices for each block. Observe that in this context, the weight of each block vertex in T is an integer no less than 2 and its degree, but the weight of each cutpoint vertex in T is precisely 1. So it is simple to enumerate all block graphs that are isomorphic to the given BC-tree T.

In short, for the second phase of the framework, let T represent a BC-tree enumerated in the first phase. For a specified weight w, our objective is to enumerate all weighted BC-trees in suit T having a total weight of w. Since T is provided, we can determine the number of vertices for the corresponding graph using Lemma 8. By constraining the weight of cutpoint

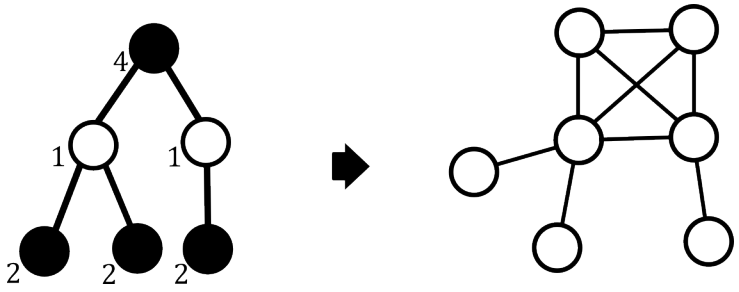

Figure 5.1: An illustration for the correspondence between a weighted BC-tree and a block graph. The number on the left-bottom of each vertex in the weighted BC-tree is the weight of the corresponding vertex. Note that in the weighted BC-tree, each cutpoint vertex has weight 1, and the degree of each block vertex should be not less than its weight, which is at least 2.

vertices to 1 and the weight of block vertices to an integer not less than 2 and its degree, each result holds a one-to-one correspondence with block graphs, which contain $w + |E(T)|$ vertices.

5.1.2 Enumeration on Cactus Graphs

Then we focus on the class of cactus graphs. A graph G is classified as a cactus graph if each edge belongs to no more than one cycle. It is, in addition, characterized as a connected graph in which each block is either an edge or a cycle. A cactus graph can be seen as a combination of many cycle graphs. In particular, a cycle graph with two vertices is denoted as K_2, while a cycle graph with one vertex is represented as K_1.

Unlike block graphs, a weighted BC-tree may correspond to multiple cactus graphs, depending on the arrangement of cutpoints within each block. Figure 5.2 shows the weighted BC-tree associated with two unisomorphic cactus graphs. As a result, we require not just the number of vertices but also the arrangement of cutpoints for each block to reconstruct a specific cactus graph. From an enumeration perspective, the arrangement of cutpoints within a block is also necessary in the second phase of the framework.

5.1.3 Enumeration on 3-leaf power graphs

Here we introduce how the framework works on the class of 3-leaf power graphs. A connected graph is a 3-leaf power if and only if it results from a tree by replacing every vertex by a clique of arbitrary size. Many studies are related to the class of 3-leaf power graphs [9, 6, 22]. In particular, Gioan and

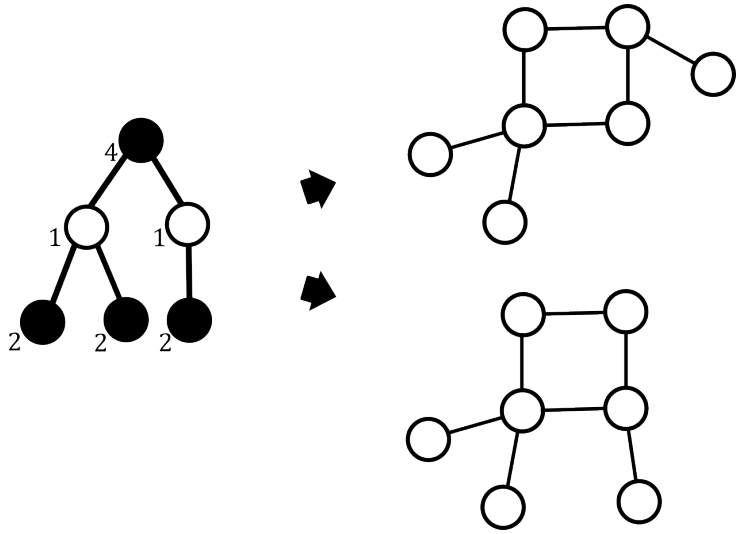

Figure 5.2: An illustration for the correspondence between a weighted BC-tree and two cactus graphs.

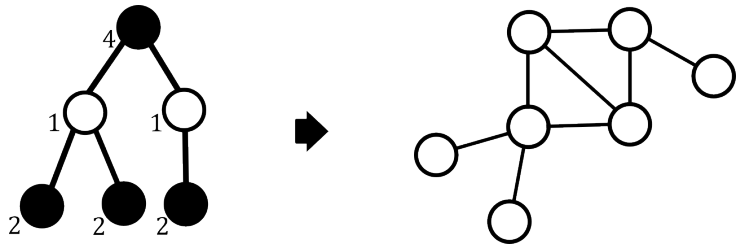

Figure 5.3: An illustration for the correspondence between a weighted BC-tree and a 3-leaf power graph.

Paul present a set of generation rules for the class of 3-leaf power graphs in [11]. A 3-leaf power graph can be constructed by (1) adding a certain number of vertices and (2) obtaining strong twins in a certain number. Yamazaki et al. proposed an enumeration algorithm for the class of 3-leaf power graphs based on this generation rule set, which enumerates 3-leaf power graphs in $O(n^3)$ time for each graph.

According to the generation rules of 3-leaf power graphs, it is obvious that a cutpoint in a 3-leaf power graph G is unable to have strong twins. Specifically, for a block $B = \{V_B, E_B\}$ in G, the number of vertices having strong twins should be $|V_B| - c$, where c denotes the number of cutpoints in B of G. Figure 5.3 illustrates that the block of 4 vertices has just a pair of strong twins, while the other two vertices serve as cutpoints.

Using the BC-tree to import the enumeration framework allows for the recursive resolution of the enumeration problem concerning 3-leaf power graphs through its subproblem, the enumeration problem of biconnected 3-leaf power graphs, which reduces the time required for enumeration. On the other hand, adding the constraint of connectedness significantly reduces the solution compared to the original problem. In conclusion, an efficient enumeration approach for the class of 3-leaf power graphs is anticipated.

Chapter 6

Conclusion and Future works

This dissertation addresses the enumeration problem for two fundamental graph structures: block-cutpoint trees and rooted vertex-weighted trees. These structures possess theoretical importance and serve as the foundations for more complex graph classes that arise in practical applications, including bioinformatics, network analysis, and data modeling.

We propose two efficient enumeration algorithms for both ordered BC-trees and unordered BC-trees, which ensure constant time delay for each output. This outcome is significant, as BC-trees contain the division of connected graphs into components and cutpoints, and their enumeration establishes the structural basis for numerous graph classes. Our approach uses the reverse search technique to prevent duplication and guarantee scalability, even with substantial input quantities.

Next, we introduced the enumeration of rooted vertex-weighted trees subjected to both structural and weight-sum constraints. Weighted trees represent cases in which vertices possess functional weights, which are common in areas such as hierarchical or biological data. We proposed an algorithm for constant amortized time delay enumeration that efficiently enumerates all such trees under specified constraints. The algorithm fits easily into the second phase of our framework, thereby improving its applicability to graph enumeration.

Based on the enumeration of BC-trees and weighted trees, we provided the notion of an extensive two-phase framework for graph enumeration. In the first phase, BC-trees are enumerated as abstract graph skeletons. During the second phase, each BC-tree is enhanced into complete graph instances through the addition of essential block-level data, such as vertex numbers. This approach extends the enumeration process, allowing its use to other graph classes, such as block graphs, cactus graphs, and 3-leaf power graphs. The framework distinguishes structure enumeration from detail instantiation,

making it both extensible and flexible.

Our research provides a cohesive viewpoint on graph enumeration by connecting tree-based structures with broader graph classes. The suggested algorithms and framework give both theoretical insights and practical tools for algorithm design and data manufacture.

There are also some promising directions related to this research.

First, in the context of enumerating weighted trees, a possible enhancement involves improving the enumeration of weight sequences. At present, our technique achieves constant amortized time for the enumeration of all weighted trees with a given tree structure and a total weight. By applying Gray code in the enumeration of weight sequences, we contend that it is possible to achieve true constant time for each output. The addition of Gray codes, which allow minimal-change traversal through combinatorial spaces, to the enumeration of weight distributions in tree topologies should significantly improve performance while ensuring correctness. A theoretical investigation into this approach, accompanied by practical execution, could result in significant improvements in weighted tree enumeration.

Second, although we have considered the case where weights can be zero in the enumeration of weighted trees—and in this context, setting the minimum weight to 0 or 1 makes little difference in practice—setting the minimum weight to 1 aligns better with typical use cases. Therefore, we consider developing an enumeration algorithm for weighted trees with a minimum weight of 1 as one of our future research directions.

Also, the natural progression of our existing research is to change our attention from weighted trees to weighted BC-trees. Since BC-trees generalize graphs by representing the block-cutpoint structure of graphs, modifying our current enumeration algorithm of weighted trees to reflect the vertex weights in BC-trees could be considered a significant improvement. This work involves investigating the efficient assignment and transmission of weights to block and cutpoint vertices while maintaining the characteristics of different vertex types. This algorithm will also enhance the enumeration framework we proposed.

At last, the enumeration framework proposed in this research allows the design of specific enumeration algorithms for various graph classes, including block graphs and cactus graphs. Although we have introduced how this framework applies to the enumeration of these classes, future studies may concentrate on establishing specific enumeration algorithms and improving these implementations. Furthermore, exploring the framework's application to more complicated classes could yield new theoretical insights and useful tools for enumeration problems in graph theory.

Bibliography

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley series in computer science and information processing. Addison-Wesley Publishing Company, 1974.

[2] David Avis and Komei Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1):21–46, 1996.

[3] H.-J. Bandelt and H.M. Mulder. Distance-Hereditary Graphs. *Journal of Combinatorial Theory*, 41:182–208, 1986.

[4] Terry Beyer and Sandra Mitchell Hedetniemi. Constant time generation of rooted trees. *SIAM Journal on Computing*, 9(4):706–712, 1980.

[5] Andreas Brandstädt and Van Bang Le. Structure and linear time recognition of 3-leaf powers. *Information Processing Letters*, 98(4):133–138, 2006.

[6] Andreas Brandstädt and Van Bang Le. Structure and linear time recognition of 3-leaf powers. *Information Processing Letters*, 98(4):133–138, 2006.

[7] Steven Chaplick, Elad Cohen, and Juraj Stacho. Recognizing some subclasses of vertex intersection graphs of 0-bend paths in a grid. In Petr Kolman and Jan Kratochvíl, editors, *International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2011)*, pages 319–330, Berlin, Heidelberg, 2011. Lecture Notes in Computer Science Vol. 6986, Springer Berlin Heidelberg.

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. Cambridge, 4th edition, 2022.

[9] Michael Dom, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Error compensation in leaf power problems. *Algorithmica*, 44:363–381, 2006.

[10] Dale R. Fox. Block cutpoint decomposition for markovian queueing systems. *Applied Stochastic Models and Data Analysis*, 4(2):101–114, 1988.

[11] Emeric Gioan and Christophe Paul. Split decomposition and graph-labelled trees: Characterizations and fully dynamic algorithms for totally decomposable graphs. *Discrete Applied Mathematics*, 160(6):708–733, 2012.

[12] Frank Harary. A characterization of block-graphs. *Canadian Mathematical Bulletin*, 6(1):1–6, 1963.

[13] Frank Harary. *Graph Theory*. Addison-Wesley, 1969.

[14] James Korsh and Paul Lafollette. A loopless gray code for rooted trees. *ACM Trans. Algorithms*, 2(2):135–152, apr 2006.

[15] Shin-ichi Nakano. Efficient generation of plane trees. *Information Processing Letters*, 84(3):167–172, 2002.

[16] Shin-ichi Nakano, Ryuhei Uehara, and Takeaki Uno. A new approach to graph recognition and applications to distance-hereditary graphs. *Journal of Computer Science and Technology*, 24(3):517–533, May 2009.

[17] Shin-ichi Nakano and Takeaki Uno. Efficient generation of rooted trees. *National Institute for Informatics (Japan), Tech. Rep. NII-2003-005E*, 8:4–63, 2003.

[18] Shin-ichi Nakano and Takeaki Uno. Constant Time Generation of Trees with Specified Diameter. In *International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2004)*, pages 33–45. Lecture Notes in Computer Science Vol. 3353, Springer-Verlag, 2004.

[19] Shin-ichi Nakano and Takeaki Uno. Constant time generation of trees with specified diameter. In Juraj Hromkovič, Manfred Nagl, and Bernhard Westfechtel, editors, *Graph-Theoretic Concepts in Computer Science*, pages 33–45, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[20] Takashi Nakayama and Yuzuru Fujiwara. BCT Representation of Chemical Structures. *Journal of Chemical Information and Computer Sciences*, 20(1):23–28, 1980.

[21] Takashi Nakayama and Yuzuru Fujiwara. Computer representation of generic chemical structures by an extended block-cutpoint tree. *Journal of Chemical Information and Computer Sciences*, 23(2):80–87, 1983.

[22] Naomi Nishimura, Prabhakar Ragde, and Dimitrios M. Thilikos. On graph powers for leaf-labeled trees. *Journal of Algorithms*, 42(1):69–108, 2002.

[23] J. Pallo. Generating trees with n nodes and m leaves. *International Journal of Computer Mathematics*, 21(2):133–144, 1987.

[24] Keith Paton. An Algorithm for the Blocks and Cutnodes of a Graph. *Commun. ACM*, 14(7):468–475, Jul 1971.

[25] Douglas Brent West. *Introduction to Graph Theory, 2nd Edition*. Prentice Hall, 2000.

[26] Robert Alan Wright, Bruce Richmond, Andrew Odlyzko, and Brendan D. McKay. Constant time generation of free trees. *SIAM Journal on Computing*, 15(2):540–548, 1986.

[27] Kazuaki Yamazaki, Mengze Qian, and Ryuhei Uehara. Efficient Enumeration of Non-Isomorphic Distance-Hereditary Graphs and Ptolemaic Graphs. In *The 15th International Conference and Workshops on Algorithms and Computation (WALCOM 2021)*, page 284–295, Berlin, Heidelberg, 2021. Lecture Notes in Computer Science Vol. 12635, Springer-Verlag.

[28] Kazuaki Yamazaki, Toshiki Saitoh, Masashi Kiyomi, and Ryuhei Uehara. Enumeration of nonisomorphic interval graphs and nonisomorphic permutation graphs. *Theoretical Computer Science*, 806:310–322, 2020.

[29] Yu Yang, Long Li, Wenhu Wang, and Hua Wang. On bc-subtrees in multi-fan and multi-wheel graphs. *Mathematics*, 9(1), 2021.

[30] Yu Yang, Hongbo Liu, Hua Wang, and Scott Makeig. Enumeration of bc-subtrees of trees. *Theoretical Computer Science*, 580:59–74, 2015.

Publications

Journal

- **Mengze Qian** and Ryuhei Uehara. Constant time enumeration of weighted trees, Discrete Applied Mathematics, Volume 375, 2025, Pages 129-138, ISSN 0166-218X, https://doi.org/10.1016/j.dam.2025.05.043.

Conferences

- **Mengze Qian** and Ryuhei Uehara. Efficient Enumeration of Non-isomorphic Block-Cutpoint Trees, The fifth Workshop on Enumeration Problems and Applications (WEPA2022). 2022, November.
- **Mengze Qian** and Ryuhei Uehara. Efficient Enumeration of Block-Cutpoint Trees, LA-Symposium, 2025, Janurary.