Master's Thesis

Efficient Enumeration of Non-isomorphic Ptolemaic Graphs

Mengze QIAN

Supervisor Mineo KANEKO

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

09, 2021

Abstract

In Euclidean geometry, Ptolemaic inequality relates six distances by four points in the plane. For any four points A, B, C, D, Ptolemaic inequality is represented as $\overline{AC} \cdot \overline{BD} \leq \overline{AB} \cdot \overline{CD} + \overline{BC} \cdot \overline{DA}$. By Ptolemaic inequality, the characterization of Ptolemaic graphs is easy to understand. A Ptolemaic graph is a connected graph which for any four vertices u, v, w, x of G, $d(u, v)d(w, x) \leq d(u, w)d(v, x) + d(u, x)d(v, w)$ holds.

Howorka shows that the class of Ptolemaic graphs is the intersection of the classes of distance hereditary graphs and chordal graphs. Hence the Ptolemaic graphs also hold the properties of both the chordal graphs and the distance hereditary graphs. A graph is said to be chordal if every cycle of length at least 4 has a chord. The class of chordal graphs is well investigated with related massive research. On the other hand, a graph G is distance hereditary if it is connected and every induced path is isometric; that is, if the distance function in every induced subgraph of G is the same as that in G itself. The vertex incremental description is one of the ways of the characterizations of a graph class, which means, by applying vertex incremental rules that add one or several vertices each time, all graphs of a certain graph class can be obtained. The vertex incremental descriptions of the classes for both distance hereditary graphs and Ptolemaic graphs are proposed by Bandelt and Mulder in 1986.

Uehara and Uno give the clique laminar tree (CL-tree) to represent a Ptolemaic graph as a tree structure. The clique laminar tree represents laminar structure on cliques in a Ptolemaic graph. Using CL-tree, Tran and Uehara propose an enumeration algorithm of Ptolemaic graphs in 2020. However, it only shows the two phases of the algorithm and gives the polynomial upper bound between the enumeration of two Ptolemaic graphs. In 2009, the DH-tree is proposed as the tree representation of the distance hereditary graph by Nakano et al. As the class of Ptolemaic graphs is the subset of the class of distance hereditary graphs, the DH-tree can also be applied to the representation of Ptolemaic graphs. By converting the DH-tree to a string representation, the graph isomorphism of distance hereditary graphs can be solved efficiently. As one of the applications for the DH-tree, Nakano et al. also give the theoretical time complexity for enumerating distance hereditary graphs by using DH-trees, whereas no specific algorithms are given. In 2018, Yamazaki et al. proposed an enumeration framework for the graph classes, which uses reverse search as the technique to avoid duplicates and solve the graph isomorphism of the graph class efficiently. Using the framework,

Yamazaki et al. proposed a new enumeration algorithm for distance hereditary graphs, which uses the vertex incremental characterization of distance hereditary graphs. Since the class of Ptolemaic graphs holds a similar vertex incremental characterization with the class of distance hereditary graphs, we can modify the algorithm to enumerate Ptolemaic graphs.

In this paper, we focus on the enumeration algorithm for Ptolemaic graphs. We first introduce the related work. Next, we give the preliminaries, which include the vertex incremental characterizations of both distance hereditary graphs and Ptolemaic graphs, then we give the notion of the DH-tree and reverse search. By proposing the notion of a function, we give an efficient way to compute if a vertex is simplicial. Then, by modifying the enumeration algorithm from distance hereditary graphs, we give the enumeration algorithm for Ptolemaic graphs, which enumerates all Ptolemaic graphs with at most n vertices in $O(n^3)$ time for each.

Contents

List of Figures

Chapter 1

Introduction

Recently, we have to deal with huge amounts of data in data mining and machine learning. It is common to find the abstract data structure of these data and enumerate them to test if the model is feasible. For the basic structures like tree structures, they are widely investigated, whereas for more complex structures, the related research is not enough yet.

The enumeration problem is a classic problem in various areas. Enumerating all elements in a certain graph class can be useful for the application of the graph class. Sometimes, we need to enumerate all elements in a specific set and analyze them. Recent years, there has been much research on the structures in graphs in the areas of graph algorithms and graph theory. In graph theory, some graph classes have not been well investigated yet, e.g., Ptolemaic graphs. From these background, we give the enumeration algorithm for Ptolemaic graphs in this thesis.

1.1 Previous research

In [10], Howorka proposes the characterizations of distance hereditary graphs and Ptolemaic graphs. A graph is a distance hereditary graph if the distances in any connected induced subgraph are the same as that in the original graph. A Ptolemaic graph is a graph that for every four vertices, the shortest path distance between vertices obeys Ptolemaic inequality. It is proved by Howorka that the class of Ptolemaic graphs is the subset of the class of distance hereditary graphs [7], and Bandelt and Mulder give the similar vertex incremental characterizations for the classes of distance hereditary graphs and Ptolemaic graphs [8].

In 2020, Tran and Uehara proposed an enumeration algorithm for Ptolemaic graphs [2], which used CL-tree [3], a tree structure based on the lam-

inar structure of intersections of maximal cliques in Ptolemaic graphs, as the tree structure in enumeration. Their algorithm has two phases. They first enumerate all CL-tree structures and then assign vertices to construct corresponding Ptolemaic graphs. After giving the algorithm, they show the following theorem for Ptolemaic graphs [2].

Theorem 1. *Non-isomorphic Ptolemaic graphs with at most n vertices can be enumerated in polynomial time for each.*

However, in this algorithm, they do not give the specific steps for enumeration, thus the exact upper bound of the polynomial is not clear, and also the algorithm is difficult to be implemented.

A DH-tree [6] is a canonical tree structure derived from a distance hereditary graph, which is proposed by Nakano et al. As one of the applications of the DH-tree, Nakano et al. give the outline of the enumeration algorithm for distance hereditary graphs as follows.

Enumeration for distance hereditary graphs by DH-trees: For given n, it can efficiently enumerate each distance hereditary graph with at most n vertices exactly once as follows; (1) enumerate all unordered trees of n leaves such that each inner node has at least two children, (2) for each tree obtained in (1), enumerate all possible assignments of labels to all inner nodes, and (3) for each label assignment in (2), enumerate all possible choices of one child as a neck for each inner node with label 'P'.

Based on the phases above, they show the following theorem;

Theorem 2. *Distance hereditary graphs with at most n vertices can be enumerated in $O(n)$ time for each, with $O(n^2)$ space.*

Here they use the constant time generation algorithm of trees proposed by Nakano and Uno [18] to enumerate all unordered trees with n leaf nodes, and each inner node of which holds at least two child nodes in step (1). Then, they enumerate all possible assignments of labels to all inner nodes in step (2) and enumerate all possibilities of the neck-pendant combination in step (3). However, "possible assignments of labels" in step (2) is not defined certainly. Furthermore, the algorithm has not been implemented yet.

Recently, a framework of enumeration algorithm has been proposed by Yamazaki et al [4], which is based on reverse search [1]. The framework has been applied in the enumeration algorithm for interval graphs and permutation graphs [17]. Upon the framework, Yamazaki et al. also give the enumeration algorithm for distance hereditary graphs [5] by using the vertex

incremental characterization of the class of distance hereditary graphs, and they show the following theorem for the enumeration of distance hereditary graphs and give the implementation.

Theorem 3. *Distance hereditary graphs with at most n vertices can be enumerated in $O(n^3)$ time for each.*

1.2 Motivation and Results

For the enumeration algorithm for Ptolemaic graphs, there is no implemented algorithm yet. In this paper, we aim to design the efficient enumeration algorithm for Ptolemaic graphs. Yamazaki has given the enumeration algorithm for distance hereditary graphs and has implemented the algorithm recently. As the classes of distance hereditary graphs and Ptolemaic graphs have the similar vertex incremental characterizations, by modifying the enumeration algorithm for distance hereditary graphs, I propose the enumeration algorithm for Ptolemaic graphs. The enumeration algorithm for Ptolemaic graphs can enumerate all Ptolemaic graphs with at most n vertices in $O(n^3)$ for each, which is more efficient than the algorithm proposed by Tran and Uehara [2]. Yamazaki also implemented the enumeration algorithm for Ptolemaic graphs and published the experimental results on [19]. This year, we presented the enumeration algorithms for both distance hereditary graphs and Ptolemaic graphs on WALCOM 2021 [5].

Chapter 2

Preliminaries

We consider only simple graphs $G = (V, E)$ with no self-loop and multi-edges. We assume $V = \{v_0, v_1, ..., v_{n-1}\}$ for some n and $|E| = m$. The *open neighborhood* of a vertex v in a graph $G = (V, E)$ is the set $N(v) = \{u \in V \mid \{u, v\} \in E\}$, and the *close neighborhood* of a vertex v is $N(v) \bigcup \{v\}$, denoted by $N[v]$. Given a graph $G = (V, E)$, 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 $\{u, v\} \notin E$. For two vertices u and v in a given graph $G = (V, E)$, they are *adjacent* if $\{u, v\} \in E$. A vertex v in G is a *pendant* only if $|N(v)| = 1$. Given an unlabeled unrooted graph $G = (V, E)$, vertices u and v are *equivalent* in G if $N(u) = N(v)$ and $|N(x)| = |N(y)| \geq 1$.

Given a graph $G = (V, E)$, a sequence of the distinct vertices $v_0, v_1, ..., v_l$ is a *path*, denoted by $(v_0, v_1, ..., v_l)$, if $\{v_j, v_{j+1}\} \in E$ for each $0 \leq j < l$. The *length* of a path is the number l of edges on the path. For two vertices u and v, the *distance* of the vertices, denoted by $d(u, v)$, is the minimum length of the paths joining u and v. A *clique* is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent. The *degree* of a vertex v in G is $|N_G(v)|$ which is denoted by $\deg_G(v)$.

Given a graph $G = (V, E)$ and a subset U of V, the *induced subgraph* by U, denoted by $G[U]$, is the graph (U, E'), where $E' = \{\{u, v\} \mid u, v \in U$ and $\{u, v\} \in E\}$. Given a graph $G = (V, E)$ and a subset U of V, an induced connected subgraph $G[U]$ is *isometric* if the distance of pairs of vertices in $G[U]$ is the same in G. A connected graph G is *distance hereditary* if every induced path in G is isometric. A connected graph G is Ptolemaic if for any four vertices u, v, w, x of G, we have $d(u, v)d(w, x) \leq d(u, w)d(v, x) + d(u, x)d(v, w)$.

Let K_n denote the complete graph of n vertices and S_n denote the star graph with one *neck* and $n-1$ pendants adjacent to the neck. Given a graph

$G = (V, E)$, a vertex v is *simplicial* in G if $G[N(v)]$ is a clique in G. We define a *graph isomorphism* between two graphs $G_0 = (V_0, E_0)$ and $G_1 = (V_1, E_1)$ as follows. The graph G_0 is *isomorphic* to G_1 if and only if there is a one-to-one mapping $\phi: V_0 \to V_1$ such that for any pair of vertices $u, v \in V_0$, $\{u, v\} \in E_0$ if and only if $\{\phi(u), \phi(v)\} \in E_1$. We denote by $G_0 \sim G_1$ for two isomorphic graphs G_0 and G_1.

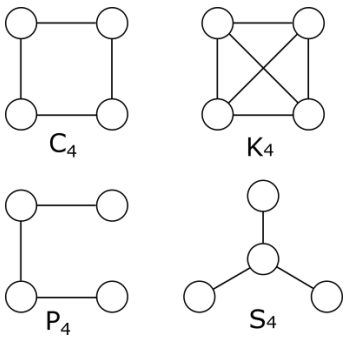

Figure 2.1: Some special graphs in graph theory with 4 vertices

2.1 Vertex Incremental Characterizations

The notion of vertex incremental characterization is proposed by Shi [11]. A *vertex incremental* characterization of a class $\mathcal{A}$ of graphs is the necessary and sufficient condition under which adding a vertex v to a graph from $\mathcal{A}$ would obtain another graph in $\mathcal{A}$. For some graph classes, adding a set of vertices to a graph from $\mathcal{A}$ is also a legal condition for the vertex incremental characterization. For the class $\mathcal{A}$ of graphs, the characterization of $\mathcal{A}$ is often written as a set of operations, and when applying either one of the operations in a specific way or in any order from a starting graph of one vertex, we would obtain all graphs in $\mathcal{A}$. There are several graph classes holding vertex incremental characterizations, parity graphs [12], 3-leaf power graphs [13], (6,2)-chordal bipartite [14], etc.

To introduce the vertices incremental description for the classes of both distance hereditary graphs and Ptolemaic graphs, we first give the notions of the following three operations for a given graph G, which are also shown in Figure 2.2.

For a vertex v in a graph $G = (V, E)$, we obtain strong twin u of v by adding a new vertex u into $V(G)$ such that $N[u] = N[v]$, in which vertices u and v are adjacent. Respectively, we obtain weak twin u of v by adding a new vertex u into $V(G)$ such that $N(u) = N(v)$, and vertices u and v are

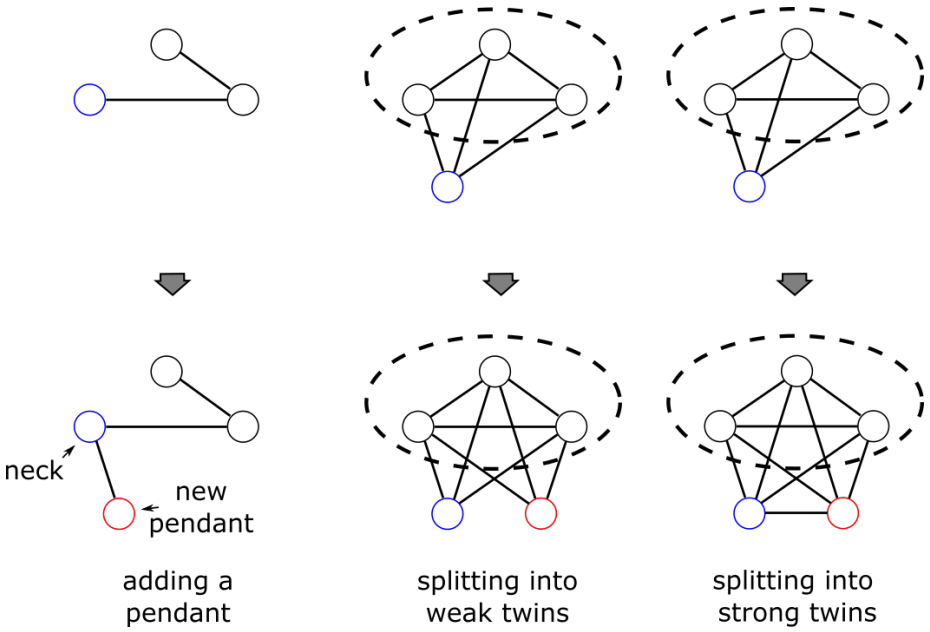

Figure 2.2: Three operations in a graph.

not adjacent. We add a pendant u on v such that u and v are adjacent and $N(u) = \{v\}$.

2.1.1 Distance Hereditary Graphs

We first give the vertex incremental characterization for the class of distance hereditary graphs. Distance hereditary graphs are first characterized by Howorka in [10], and Bandelt and Mulder [8] give the vertex incremental characterization of the distance hereditary graphs as follows;

Theorem 4 ([8, Theorem 1])**.** *A vertex is a distance hereditary graph. Let $G = (V, E)$ be a distance hereditary graph, and v be any vertex in G. Then a graph obtained by either (1) adding a pendant $u \notin V$ to v, (2) splitting v into weak twins, or (3) splitting v into strong twins, is a distance hereditary graph.*

2.1.2 Ptolemaic Graphs

For Ptolemaic graphs, we have the following characterization due to Howorka [7]:

Theorem 5. *The following conditions are equivalent: (1) G is Ptolemaic;*

(2) G is distance hereditary and chordal; (3) for all distinct non-disjoint maximal cliques P, Q of G, $P \bigcap Q$ separates $P \backslash Q$ and $Q \backslash P$.

For the vertex incremental characterization of the class of Ptolemaic graphs, we note that in [2, Corollary 1], the authors give the characterization of Ptolemaic graphs by only adding strong twins and pendants, which is incorrect. The correct one of Ptolemaic graphs should be given as follows:

Theorem 6 ([8, Corollary 6])**.** *A vertex is a Ptolemaic graph. Let $G = (V, E)$ be a Ptolemaic graph, and v be any vertex in G. Then a graph obtained by either (1) adding a pendant $u \notin V$ to v, (2) splitting v into weak twins if vertex v is simplicial, or (3) splitting v into strong twins, is a Ptolemaic graph.*

2.2 The DH-tree

2.2.1 Generation of DH-trees

We use the DH-tree to represent a Ptolemaic graph. A *DH-tree* is a canonical tree structure derived from a distance hereditary graph, which is a rooted ordered tree defined in [6]. In a DH-tree, each inner node holds a label, and the label of an inner node is one of {'S', 'W', 'P'}, which indicates strong twin, weak twin, or pendant with neck.

We define *S-root* (and *P-root*) as the root node of a DH-tree that holds 'S' label ('P' label, respectively). Similarily, we define *S-node* (and *W-node*, *P-node*) as the inner node of a DH-tree that holds 'S' label ('W' label and 'P' label, respectively).

There are two kinds of DH-trees; one is directly derived from a distance hereditary graph, and the other is the *normalized* DH-tree obtained from a DH-tree by contracting parent and child node pairs that hold the same labels.

First we introduce the DH-tree $\mathcal{T}$ derived from a distance hereditary graph G. Since K_1 is the distance hereditary graph with only one vertex, and K_2 is the distance hereditary graph that can be obtained from K_1 by either adding a pendant or adding a strong twin, we define K_1 and K_2 as the special distance hereditary graphs. Then, we consider the distance hereditary graphs with at least 3 vertices. For a given distance hereditary graph G, we define three families of vertex sets in G as follows;

$\mathcal{S} = \{S \mid x, y \in S$ if $N[x] = N[y]$ and $|S| \geq 2\}$
$\mathcal{W} = \{W \mid x, y \in W$ if $N(x) = N(y), |S| \geq 2$, and $|N(x)| = |N(y)| > 1\}$
$\mathcal{P} = \{P \mid x, y \in P$ if x is a pendant and y is its neck $\}$

Then, we have the following lemma.

Lemma 7 ([6, Lemma 4]). *(1) Each set $P \in \mathcal{P}$ contains exactly one neck with associated pendants. (2) Each $v \in V$ belongs to either (a) exactly one set in $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P}$, or (b) no set in the families. (3) For any distance hereditary graph G with at least 3 vertices, $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} \neq \emptyset$.*

The DH-tree is a tree structure derived from a distance hereditary graph, which is generated from leaf node to root. Here, we first consider three basic cases of distance hereditary graphs G and their corresponding DH-trees $\mathcal{T}$.

Case 1: G is K_1.

In this case, DH-tree $\mathcal{T}$ is a single root with no label.

Case 2: G is K_n when $n \geq 2$.

In this case, DH-tree $\mathcal{T}$ holds a S-root and n leaf nodes with no labels.

Case 3: G is S_n when $n \geq 3$.

In this case, DH-tree $\mathcal{T}$ holds a P-root and n leaf nodes with no labels. For a star graph with n vertices, it holds a neck (center) and $n-1$ pendants. Here $n-1$ pendants are equivalent since we consider the star graph as an unlabeled graph, and all pendants are adjacent with the neck. To distinguish the pendants and the neck, we define $L(u)$ as the set of child nodes of inner node u in $\mathcal{T}$. In the Case 3, u is the root of $\mathcal{T}$ and $v \in L(u)$ is the leaf node of $\mathcal{T}$. We also define w as the leftmost child node of u in $\mathcal{T}$, which corresponds to the neck, and the remaining $x \in L(u) \setminus \{w\}$ correspond to the pendants.

Note that here n is the number of vertices in G, and each leaf node in DH-tree $\mathcal{T}$ corresponds to a vertex in G. Also, K_2 is defined as a clique not a star, which belongs to Case 2.

Next, we define the DH-tree $\mathcal{T}$ with n vertices in general case, where $n \geq 3$. For the general case of G, we assume that G is neither K_n nor S_n.

Since each leaf node in $\mathcal{T}$ corresponds to a vertex in G, and the DH-tree is a tree structure obtained from leaf node to root node, we can group all leaf nodes in $\mathcal{T}$ by three kinds of vertex set $\mathcal{S}, \mathcal{W}$ and $\mathcal{P}$. Here $\mathcal{S} \cup \mathcal{W} \cup \mathcal{P} \neq \emptyset$ based on Lemma 6. Here, for any set $S \in \mathcal{S}$ or set $W \in \mathcal{W}$, all vertices in S or W are equivalent. For any set $S \in \mathcal{S}$, we generate an inner node with label 'S' and let inner node be the common parent of these leaf nodes. We do the same operation on all sets $S \in \mathcal{S}$. Also, for any set $W \in \mathcal{W}$, we similarily generate an inner node with label 'W' and let inner node be the common

parent of these leaf nodes. For any set $P \in \mathcal{P}$, we generate an inner node with label 'P' and let inner node be the common parent of these leaf nodes. However, pendant case includes one neck and several pendants. Therefore, same as the Case 3 of the basic cases above, we define $L(u)$ as the set of child nodes of inner node u in $\mathcal{T}$ and w as the leftmost child node of u in $\mathcal{T}$. Then, w corresponds to the neck, and the remaining $x \in L(u) \setminus \{w\}$ correspond to the pendants.

After generating S-node in DH-trees, we contract each vertex set of S into one of the strong twins in S. Similarily, we contract each vertex set of W into one of the weak twins in W after generating W-node. For each P, we generate P-node in $\mathcal{T}$ and contract all pendants into neck in G.

An execution of the generation of DH-trees is shown in Figure 2.3.

By repeating the process until the resultant graph becomes one of the basic cases, we generate the root node of $\mathcal{T}$ and obtain $\mathcal{T}$.

2.2.2 Normalization of DH-trees

In this section, we introduce the normalization of DH-trees.

A DH-tree $\mathcal{T}$ derived from a distance hereditary graph can be compacted in some cases. Here, recall that in the vertex incremental characterization of distance hereditary graphs, we add on vertex in each step to obtain the distance hereditary graph G with n vertices. For two generation operation orders of a distance hereditary graph, 1) first splitting vertex u into weak twins u and v, then splitting vertex v into weak twins v and w; 2) splitting vertex u into weak twins u, v and w, we obtain the same graph. The same reduction can also be applied to the S-nodes. For pendant case, we consider two generation operation orders as follows, 1) first adding a pendant v on neck u, then adding a pendant w on neck u; 2) adding pendants v and w on u. Also, we obtain the same graph. From this notion, we here consider the normalization of DH-trees.

We assume a S-node/W-node v in $\mathcal{T}$ holds the same label as its parent node v', then for the child nodes of both v and v', they can be the weak/strong siblings. Also we assume a P-node v in $\mathcal{T}$ holds the same label as its parent node v', and v is the leftmost child node of v' in $\mathcal{T}$, then for the child nodes of both v and v', they can be the pendants on the same neck. Here the only one exception is that the leftmost child node of v is the neck of all pendants. Hence we keep the leftmost child node of v as the leftmost child node after normalizing. Figure 2.4 shows an example for the normalization of the DH-tree given in Figure 2.3.

Here, normalization can be applied in the *depth first* mannar [16], and the normalized DH-tree $\mathcal{T}'$ can be obtained from $\mathcal{T}$ in $O(n)$ time and space.

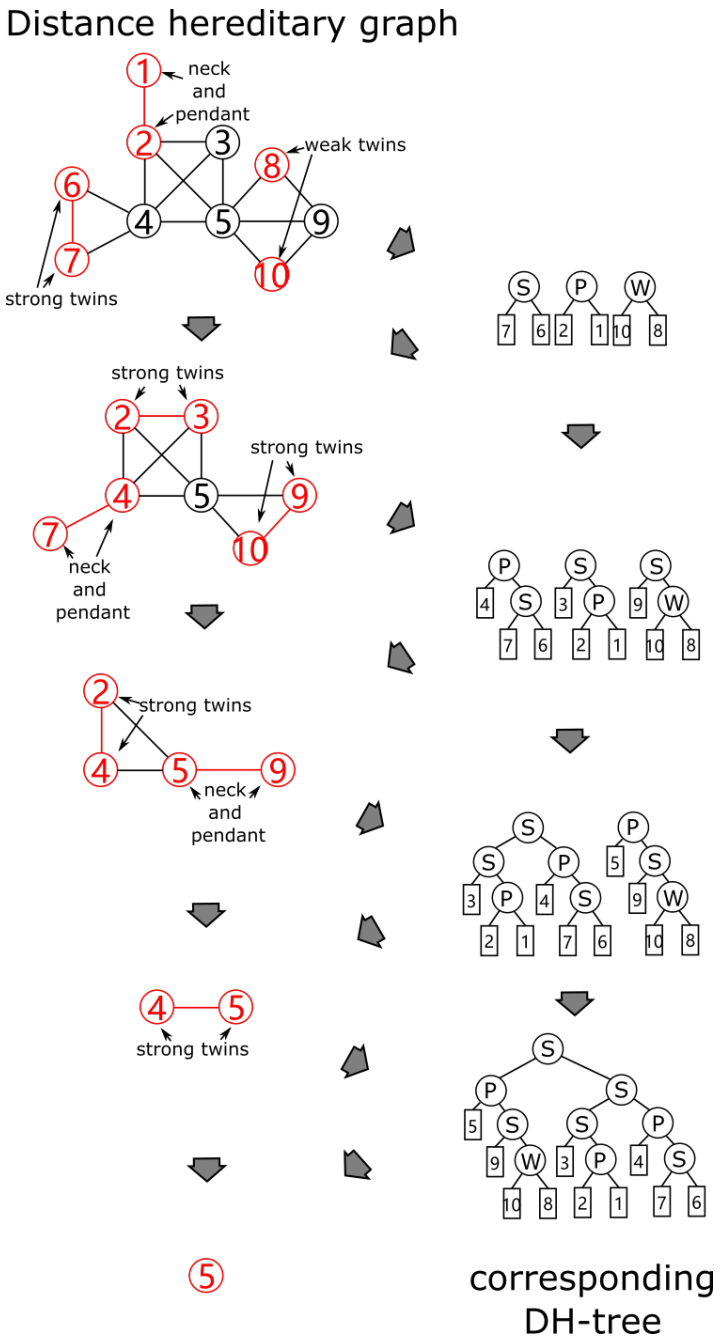

Figure 2.3: Generating the DH-tree by compacting vertices in a distance hereditary graph.

For the normalized DH-tree, the following theorem is given:

Theorem 8 ([6, Theorem 4])**.** *The normalized DH-tree of a connected dis-*

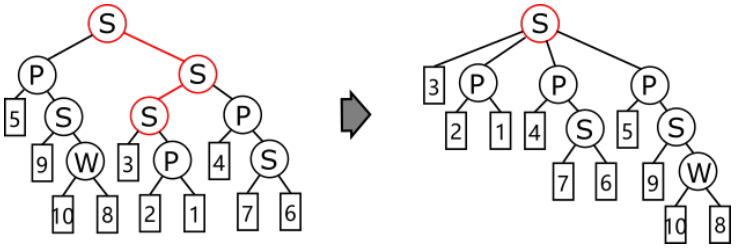

Figure 2.4: Normalization of the DH-tree in Figure 2.3.

tance hereditary graph is canonical, which means that the normalized DH-tree $\mathcal{T}$ for a connected distance hereditary graph G is uniquely determined, and the original distance hereditary graph G is uniquely constructed from the normalized DH-tree $\mathcal{T}$.

Based on the definitions above, the DH-tree is almost an ordered tree. We define the DH-tree $\mathcal{T}$ as a *right-heavy* tree, that is, for the inner node p in $\mathcal{T}$, its child nodes q are ordered by right-heavy manner. For two child nodes q and q' of p, if $\mathcal{T}_q$ is greater than $\mathcal{T}_{q'}$ (here $\mathcal{T}_q$ indicates the subtree with root q in $\mathcal{T}$, $\mathcal{T}_{q'}$ respectively), we let child node q lay on the right side of q'. We compare two subtrees by height from right to left (we define the *height* of a DH-tree in a natural way. The leaf node has height 0, and the other node has the maximal height of their subtrees). To give the unique tree representation of a distance hereditary graph, we need to give an order among inner nodes in $\mathcal{T}$. Here we define the priority order among inner nodes as: P-node $>$ W-node $>$ S-node. For the case when two subtrees hold the same structure, we order two subtrees by the order among inner nodes.

Similar to the isomorphic of graphs, we assume the DH-tree $\mathcal{T}_0$ is *isomorphic* to $\mathcal{T}_1$ if $\mathcal{T}_0$ and $\mathcal{T}_1$ hold the same tree structure and the same layout, denoted by $\mathcal{T}_0 \sim \mathcal{T}_1$.

2.2.3 Three Graph Representations

We use the *adjacency list* [16] as the standard graph representation, which is consisted of finite unordered lists for each vertex. The elements in the list represent to the vertices that are adjacent to each vertex.

Convert Between Adjacency List and DH-trees

In [6], Nakano et al. use the notion of prefix trees, which are also called *tries* [15], of open and closed neighbors. DH-trees can be obtained efficiently by using the open and closed prefix trees of a distance hereditary graph. By

using open and closed prefix trees of a distance hereditary graph, Nakano et al. give the following theorem for the generation of the DH-tree.

Theorem 9 ([6, Theorem 6])**.** *If G is a distance hereditary graph, the DH-tree $\mathcal{T}$ derived from G can be constructed in $O(|V| + |E|)$ time and space.*

Based on Theorem 9, by using two prefix trees of a G, we can convert G into DH-tree $\mathcal{T}$ in $O(n + m)$ time, where m is the number of edges and n is the number of vertices in G.

For converting the DH-tree to the adjacency list, since each inner node in the DH-tree corresponds to an operation for obtaining the distance hereditary graph, we can obtain the distance hereditary graph by applying each inner node as an operation in the DH-tree. Both DFS and BFS are acceptable since it is ordered between parent node and child node in the DH-tree, whereas among the generation of sibling nodes, any order is available. Since we consider the distance hereditary graph $G = (V, E)$, the time complexity should also be $O(n + m)$ time.

Convert Between DH-trees and String Representation

Here we give the notion for string representation of the DH-tree.

By applying the pre-order traversal on the DH-tree, we can convert the DH-tree into its string representation. Since a distance hereditary graph corresponds to a unique DH-tree, and a DH-tree also corresponds to a unique string representation of it. The string representation of the DH-tree can be used to solve the graph isomorphism between two distance hereditary graphs and avoid the duplicants when enumerating. The pre-order traversal of a DH-tree can be applied in the depth first mannar.

Given a DH-tree $\mathcal{T}$, for the visiting inner nodes when traversing $\mathcal{T}$, we use string 'S()'(and 'W()', 'P()') to represent S-node (W-node, P-node, respectively). On the other hand, for the leaf nodes in $\mathcal{T}$ with no label, we use string 'L' to represent. Traversing $\mathcal{T}$ by depth first mannar, each string corresponding to a subtree is written between '(' and ')' of the string of their parent nodes, and the order of all strings of subtrees that hold the same parent node is obeying the order of all subtrees in $\mathcal{T}$. Therefore, converting the DH-tree to its string representation takes $O(n)$ time and vice versa.

Figure 2.5 shows an example for a DH-tree and its string representation.

2.3 Reverse Search

Reverse search is an efficient technique when dealing with enumeration problems, which is proposed by Avis and Fukuda [1]. When enumerating, each

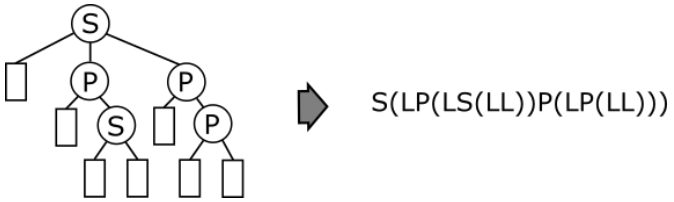

Figure 2.5: A DH-tree and its string representation

element should be enumerated only once, which means, as an efficient enumeration algorithm, it is necessary to check if the element has been enumerated or not. We define the family tree of reverse search, in which each inner node corresponds to an element that needs to be enumerated. If we keep all elements in a family tree in memory, it is easy to enumerate every element in the family tree by traversing the family tree. However, considering the space cost and time efficiency, we need the strategy when traversing the family tree.

In the context of enumeration for Ptolemaic graphs, we define the *family tree* $\hat{\mathcal{T}} = (\hat{V}, \hat{E})$ as follows. Each element in $\hat{V}$ is a distance hereditary graph G, and each arc (G_1, G_2) in $\hat{E}$ joins G_1 and G_2 if G_1 is the *parent* of G_2. In the family tree, we define the *depth* of root node as 0, and the other nodes have depth $d+1$ where d is the depth of its parent node. Here, let $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ with $|V_1| = |V_2| - 1$. We denote $\mathcal{T}_1$ as the DH-tree of G_1 and $\mathcal{T}_2$ as the DH-tree of G_2. By applying DFS on $\mathcal{T}_2$, we find the first inner node v having only leaf nodes. Then we remove the vertex which corresponds to the rightmost child node of v from G_2, denoted by G_2'. Then in family tree $\hat{\mathcal{T}}$, G_1 is the (unique) parent of G_2 if and only if $G_1 \sim G_2'$. Here, we denote G_2' as *parent(G_2)*. Thus we have $G_1 \sim \text{parent}(G_2)$. Note that we define each element in family tree $\hat{\mathcal{T}}$ as a distance hereditary graph, and each distance hereditary graph also corresponds to a unique DH-tree. Hereafter, we also use $\mathcal{T}_1 \sim \mathit{parent}(\mathcal{T}_2)$ in some contexts, which holds the same meaning as $G_1 \sim \mathit{parent}(G_2)$.

As we have shown before, each child graph contains one more vertex than its parent in $\hat{\mathcal{T}}$. After defining the parent-child relationship, the traversal of family tree starts from root by breadth first manner. Here we define K_2 as the root of family tree since K_1 and K_2 are output first as the special graphs. Using the family tree, all distance hereditary graphs with n vertices are obtained from the distance hereditary graphs with $n-1$ vertices, and we can enumerate all non-isomorphic distance hereditary graphs of up to n vertices by traversing all elements of depth $n-2$ in the family tree.

Chapter 3

Simplicial Function

Considering the difference between the vertex incremental characterizations of the classes of distance hereditary graphs and Ptolemaic graphs, for Ptolemaic graphs, we are able to generate weak twins only if the vertex is simplicial. Thus, to check if a vertex is simplicial efficiently, we introduce the notion of a *simplicial function*.

Given a graph $G = (V, E)$, the input of simplicial function $s(v)$ is a vertex $v \in V$, and output $s(v) = true$ if the vertex v is simplicial in G, $s(v) = false$ otherwise.

Based on the vertex incremental characterization of Ptolemaic graphs, we consider how the simplicial function is updated in the following three cases. Here, we consider the Ptolemaic graph $G = (V, E)$ with $|V| \geq 2$. For K_2, since both two vertices are simplicial, let u and v be the two vertices in K_2, then we initial the simplicial function of u and v as $s(u) = s(v) = true$.

- **Pendant.** We consider adding a pendant v on a neck u, then for the pendant v, $s(v) = true$, and for the neck u, $s(u) = false$. Here, for each pendant v, we have $N(v) = \{u\}$, and for the neck u, $|N(u)| \geq 1$ and every two neighbors in $N(u)$ are not adjacent. It is easy to see that for other vertices $w \in G$, $s(w)$ is not changed. One special case for pendant is when we add a new pendant on K_1, whereas both two vertices in K_2 are simplicial. We start our enumeration algorithm from K_2 to avoid this special case. This step requires $O(1)$ time to update.

- **Strong twins.** We consider splitting a vertex $v \in V$ into the strong twins u and v with $N[u] = N[v]$, which means we also need to add an edge $\{u, v\}$ into E. Since here vertices u and v are equivalent, for u and v, $N[v] = N[u]$, we have $s(v) = s(u)$. For $w \in V \setminus \{u, v\}$, same as the pendant case above, $s(w)$ is not changed in this case. Therefore, we only need to set $s(u) = s(v)$ in $O(1)$ time.

- **Weak twins.** We consider splitting a vertex $v \in V$ into the weak twins u and v with $N(u) = N(v)$. Since for both strong twins and weak twins, we have $N(u) = N(v)$, we also have $s(u) = s(v)$. On the other hand, for $w \in N(u)$, since vertices u and v are not adjacent, and both u and v are the neighbors of w, we have $s(w) = false$. For $x \in V \setminus N[v]$, $s(x)$ is not changed. Thus we need to set $s(u) = s(v)$, and for each $w \in N(v)$, $s(w) = false$. This step takes $O(\deg_G(v))$ time.

Based on the case analysis above, we have the following lemma:

Lemma 10. *For each generation rule (1), (2), and (3) in Theorem 6, the update of the function $s(v)$ of each vertex v can be done in $O(n)$ time, where $n = |V|$.*

Chapter 4

Enumeration Algorithm

In this section, we give the enumeration algorithm of Ptolemaic graphs.

Based on Lemma 10, we can compute the simplicial function of all vertices in a given normalized DH-tree. Using the same framework of enumeration algorithm of distance hereditary graphs, we give Algorithm 1 as the outline

of the enumeration algorithm for Ptolemaic graphs.

Algorithm 1: Outline of Enumeration Algorithm for Ptolemaic graphs

```
1  Input: An integer n
2  Output: All Ptolemaic graphs with at most n vertices
3  output K_1;
4  S ← {T(K_2)};
5  while S ≠ ∅ do
6  |  get DH-tree T from S;
7  |  convert T into adjacency list of G;
8  |  if |V(G)| ≤ n then
9  |  |  output G;
10 |  |  convert G into T;
11 |  |  generate CH(T) by T;
12 |  |  for T' ∈ CH(G) do
13 |  |  |  find the leftmost inner node v having only leaf nodes in T'
   |  |  |   by DFS;
14 |  |  |  convert T' into G';
15 |  |  |  generate parent(G') by removing the vertex in G'
   |  |  |   corresponding to the rightmost leaf node of v;
16 |  |  |  convert parent(G') into parent(T');
17 |  |  |  if T ~ parent(T') then
18 |  |  |  |  push T' into S;
19 |  |  |  end
20 |  |  end
21 |  end
22 end
```

In Algorithm 1, the input integer n corresponds to the maximum number of vertices for Ptolemaic graphs, and the outputs are all non-isomorphic Ptolemaic graphs with at most n vertices.

Here we show the time complexity of Algorithm 1. Comment after line number in each line indicates the time complexity for each step. M indicates the number of the graphs with at most n vertices.

- **Line 3 ($O(1)$ time)** We consider K_1 and K_2 as the special cases of our enumeration, hence we output K_1 first as one of the Ptolemaic graphs.
- **Line 4 ($O(1)$ time)** Initialize set $\mathcal{S}$ by $\mathcal{T}(K_2)$, the DH-tree of K_2. Here $\mathcal{S}$ is the set of the DH-tree that can be outputted. Note that here we standardize K_2 as the graph obtained from K_1 by adding a strong twin. Initialization of set $\mathcal{S}$ takes $O(1)$ time.

- **Loop 5-22 ($O(M)$ time for iteration, $O(n^3)$ time for each loop)** Check if $\mathcal{S}$ is empty.
  - **Line 6 ($O(n)$ time)** Pop one element out from $\mathcal{S}$ and remove it from $\mathcal{S}$
  - **Line 7 ($O(n+m)$ time)** Convert the DH-tree $\mathcal{T}$ into adjacency list G, which takes $O(n+m)$ time since G contains n vertices and m edges.
  - **Line 8 ($O(1)$ time)** Check if the number of vertices of G is less equal than n. Note that the set $\mathcal{S}$ should be a queue, since the number of vertices for all graphs in $\mathcal{S}$ is ordered.
  - **Line 9 ($O(n+m)$ time)** Output G as a Ptolemaic graph, since G contains n vertices and m edges.
  - **Line 10 ($O(n^2)$ time)** Convert G from adjacency list to DH-tree $\mathcal{T}$.
  - **Line 11 ($O(n^3)$ time)** Obtain a set of DH-tree $CH(\mathcal{T})$ that each $\mathcal{T}' \in CH(\mathcal{T})$ corresponds to a G'. Here G' is the graph obtained from G by applying adding one pendant, weak twin or strong twin. Details of line 11 are shown in Algorithm 2.
  - **Loop 12-20 ($O(n)$ time for iteration, $O(n^2)$ time for each loop)** Loop in $CH(\mathcal{T})$ takes $O(n)$ time, since $|CH(\mathcal{T})| = O(n)$.
    - = **Line 13 ($O(n)$ time)** By using DFS, we find the leftmost inner node v in $\mathcal{T}'$ that holds only leaf nodes.
    - = **Line 14 ($O(n^2)$ time)** Convert DH-tree $\mathcal{T}'$ into adjacency list G'.
    - = **Line 15 ($O(n)$ time)** Remove the corresponding vertex of the rightmost leaf node of v in the adjacency list of G' to obtain $parent(G')$, since all neighbors of v in G need to be updated once.
    - = **Line 16 ($O(n^2)$ time)** Generate $parent(\mathcal{T}')$ from $parent(G')$.
    - = **Line 17 ($O(n)$ time)** Check if $\mathcal{T}$ is isomorphic to $parent(\mathcal{T}')$. To figure out if two DH-trees are isomorphic, we traverse two DH-trees to check if the tree structure and the layout are the same. Traversing two trees takes $O(n)$ time.
    - = **Line 18 ($O(1)$ time)** Push $\mathcal{T}'$ into $\mathcal{S}$ as $\mathcal{T}'$ is the DH-tree obtained from $\mathcal{T}$ based on the unique parent-child relationship in family tree we defined.

For the generation of the set $CH(\mathcal{T})$ of $\mathcal{T}$ in the line 11 of Algorithm 1, we have:

Algorithm 2: Outline for generating $CH(\mathcal{T})$ of $\mathcal{T}$

```
1  Input: T. The DH-tree T which corresponds to the Ptolemaic
     graph G.
2  Output: CH(T). CH(T) is the DH-tree set of all Ptolemaic
     graphs G' obtained from G by appling one of the vertex
     incremental rules.
3  convert T into G;
4  for v ∈ V(G) do
5  |  obtain G_p from G by adding a pendant u on neck v;
6  |  compute s(u) in G_p and update s(v);
7  |  obtain G_s from G by splitting v into strong twins u and v ;
8  |  set s(u) = s(v) in G_s;
9  |  if s(v) = true then
10 |  |  obtain G_w from G by splitting v into weak twins u and v ;
11 |  |  set s(u) = s(v) and update s(w) for w ∈ N(v) in G_w;
12 |  end
13 |  convert G_p, G_s and G_w into T_p, T_s and T_w;
14 |  convert T_p, T_s and T_w into string representations and insert into a
   |   trie;
15 end
16 extract all strings from trie;
17 for all strings extracted from trie do
18 |  convert into T' and insert T' into CH(T);
19 end
```

For Algorithm 2, the input of the algorithm is the DH-tree $\mathcal{T}$ which represents the Ptolemaic graph G. The output is the set $CH(\mathcal{T})$ that each $\mathcal{T}' \in CH(\mathcal{T})$ corresponds to the Ptolemaic graph G', which can be obtained from G by applying one of the vertex incremental rules. Elements in $CH(\mathcal{T})$ are up to isomorphism.

Here we discuss the time complexity of Algorithm 2. Same as the discussion of Algorithm 1, comment after line number in each line indicates the time complexity of the step.

- **Line 3 ($O(n+m)$ time)** Convert the DH-tree into the adjacency list.
- **Loop 4-15 ($O(n)$ time for iteration, $O(n^2)$ time for each loop)** Loop in $V(G)$ takes $O(n)$ time, since we have $|V(G)| = O(n)$ for set $V(G)$.

- **Line 5 ($O(n+m)$ time)** Copying G_p from G takes $O(n+m)$ time. Here $m = |E|$ and $n = |V|$. Then adding a new vertex on G_p takes $O(1)$ time.
- **Line 6 ($O(1)$ time)** For the pendant u in G_p, we have $N(u) = \{v\}$. Thus we set $s(u) = true$. As we consider the Ptolemaic graphs with at least 3 vertices, for the neck v, we have $\{u, w\} \notin E(G_p)$ for $w \in N(v)$. Therefore, we set $s(v) = false$. Setting two simplicial functions takes $O(1)$ time.
- **Line 7 ($O(n+m)$ time)** Same as line 5, copying G_s from G takes $O(n+m)$ time. For splitting u and v as strong twins where $u \notin V(G)$, copying the edges of v to u and making u adjacent to v take $O(n)$ time since $N[u] = N[v]$.
- **Line 8 ($O(1)$ time)** Since $N[u] = N[v]$, we set $s(u) = s(v)$ in G_s, which takes $O(1)$ time.
- **Line 9 ($O(1)$ time)** Check the simplicial function of vertex v. As we update the simplicial function of each vertex in enumeration, here it only takes $O(1)$ time.
- **Line 10 ($O(n+m)$ time)** Copying G_w from G takes $O(n+m)$ time. For splitting u and v as weak twins where $u \notin V(G)$, copying the edges of v to u takes $O(n)$ time since $N(u) = N(v)$.
- **Line 11 ($O(n)$ time)** In G_w, since $N(u) = N(v)$, we set $s(u) = s(v)$. For $w \in N(v)$, we set $s(w) = false$ since $\{u, v\} \notin E(G')$. Updating all neighbors of v takes $O(n)$ time.
- **Line 13 ($O(n+m)$ time)** We consider G_p, G_s and G_w obtained in line 6, 7 and 9. Converting Ptolemaic graph from adjacency list to DH-trees takes $O(n+m)$ each.
- **Line 14 ($O(n)$ time)** Converting $\mathcal{T}_p$, $\mathcal{T}_s$ and $\mathcal{T}_w$ generated in line 13 from DH-trees to string representations takes $O(n)$ each. Inserting all string representations into a trie also takes $O(n)$ time each.

- **Line 16 ($O(n)$ time)** In the loop 4-15, isomorphic distance hereditary graphs may be obtained repeatedly. By inserting all strings of new obtained G_p, G_s or G_w into a trie and extracting them, each string representation of $\mathcal{T}'$ can be extracted only once. Namely, we extract all strings from trie to avoid duplicates. Converting from the adjacency list to the DH-tree takes $O(n+m)$ each.

- **Loop 17-19 ($O(n)$ time for iteration, $O(n)$ time for each loop**

– **Line 16 ($O(n)$ time)** For each string extracted from trie in line 14, convert it into a DH-tree. It takes $O(n)$ time. Then push the generated DH-tree into the set $CH(\mathcal{T})$. It takes $O(1)$ time.

Based on the introduction of the simplicial function in Chapter 3, it takes $O(1)$ time to update the case when adding a pendant and splitting strong twins, and $O(n)$ time to update the case when splitting weak twins due to the update of sibling nodes. Thus we can update the simplicial function for the new vertices in $O(n)$ time.

For the total algorithm, it enumerates M graphs with at most n vertices in $O(Mn^3)$ time, for each output, it takes $O(n^3)$ time, which is shown in the Loop 5-22 of Algorithm 1. Hence, we have the following theorem.

Theorem 11. *Ptolemaic graphs with at most n vertices can be enumerated in $O(n^3)$ time for each.*

Chapter 5

Conclusion

In Chapter 1, we gave the introduction of the enumeration problem for Ptolemaic graphs. Since the efficient enumeration algorithm of Ptolemaic graphs has not been proposed yet. In this paper we gave the enumeration algorithm for Ptolemaic graphs, which is modified from the enumeration algorithm for distance hereditary graphs proposed recently. Then we gave the previous research related to the enumeration of Ptolemaic graphs. In 2020, Tran and Uehara proposed an enumeration algorithm based on CL-tree, which is a tree structure based on the laminar structure of the intersections of maximal cliques in Ptolemaic graphs, whereas their algorithm did not give the specific time complexity and pseudo-codes. In 2009, Nakano et al. proposed the tree representation of distance hereditary graphs, the DH-tree, and claimed distance hereditary graphs can be enumerated in $O(n^2)$ time for each as one of the applications of the DH-tree, whereas the algorithm has not been implemented yet. Recently, an enumeration algorithm for distance hereditary graphs has been proposed by Yamazaki et al., which is proved to be capable of enumerating all distance hereditary graphs correctly. Since the class of Ptolemaic graphs holds similar vertex incremental characterization with the class of distance hereditary graphs, we consider modifying it into Ptolemaic graphs by using the same enumeration framework based on reverse search.

Next, in Chapter 2, we gave the preliminaries needed in graph theory, and introduced the vertex incremental characterizations for both distance hereditary graphs and Ptolemaic graphs proposed by Howorka. We also showed the three kinds of graph representations for Ptolemaic graphs, conversion among them and the time complexity. Also we gave the notion of DH-trees, which is used as the tree representation of the Ptolemaic graphs. After that, we introduced the search technique, reverse search, which is the technique widely used in enumeration problems.

In Chapter 3, we gave the notion of simplicial function, which can com-

pute all vertices in a given Ptolemaic graph efficiently. Then in Chapter 4, based on the enumeration algorithm of distance hereditary graphs proposed recently, we modified the enumeration algorithm of Ptolemaic graphs. Since for Ptolemaic graphs, only simplicial vertices are able to obtain weak twins. Using amortized analysis, we can reduce time complexity when checking simplicial vertex from $O(n^2)$ to $O(1)$, which ensures that all Ptolemaic graphs with at most n vertices can also be enumerated in $O(n^3)$ for each. The enumeration algorithm for Ptolemaic graphs is given.

For the future work, we consider the following cases:

- Give the specific enumeration algorithm for the distance hereditary graphs proposed in [6]. More precisely, give the specific labeling rules for DH-tree. Then, modify the algorithm to Ptolemaic graphs
- Research on the Ptolemaic graphs, and give the properties of the Ptolemaic graphs.
- Solve the enumeration problem for other discrete structures by using the same pattern.

Appendix A

Experimental Results

Yamazaki has already implemented the enumeration algorithms for distance hereditary graphs for $n = 1, 2, ..., 14$ and Ptolemaic graphs for $n = 1, 2, ..., 15$ [5]. Here I summarize the number of Ptolemaic graphs for $n = 1, 2, ..., 15$ based on the experimental results. The full experimental results are published on [19].

Enumeration for Ptolemaic graphs	
Number of vertices	Number of Ptolemaic graphs
1	1
2	1
3	2
4	5
5	14
6	47
7	170
8	676
9	2834
10	12471
11	56675
12	264906
13	1264851
14	6150187
15	30357300

Bibliography

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

[2] Dat H. Tran and Ryuhei Uehara. Efficient enumeration of non-isomorphic ptolemaic graphs. In *The 14th International Conference and Workshops on Algorithms and Computation (WALCOM 2020)*, pages 296–307. Lecture Notes in Computer Science Vol. 12049, 2020.

[3] Ryuhei Uehara and Yushi Uno. Laminar structure of ptolemaic graphs with applications. *Discrete Applied Mathematics*, 157(7):1533–1543, 2009.

[4] Kazuaki Yamazaki, Toshiki Saitoh, Masashi Kiyomi, and Ryuhei Uehara. Enumeration of nonisomorphic interval graphs and nonisomorphic permutation graphs. *Theoretical Computer Science*, 806:323–331, January 2020.

[5] 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)* , pages 285–295. Lecture Notes in Computer Science Vol. 12635, 2021.

[6] Shin-ichi Nakano, Ryuhei Uehara and Takeaki Uno, A new approach to graph recognition and applications to distance-hereditary graphs. *J. Comput. Sci. Technol.* **24**, 517–533, 2009.

[7] Edward Howorka. A characterization of ptolemaic graphs. *Journal of Graph Theory*, 5:323–331, 1981.

[8] Hans-Jürgen Bandelt and Henry M. Mulder. Distance-Hereditary graphs. *Journal of Combinatorial Theory, Series B*, 41(2):182–208, October 1986.

[9] Maryam Bahrani and Jérémie Lumbroso. Enumerations, Forbidden Subgraph Characterizations, and the Split-Decomposition. *ArXiv abs*/1608.01465, 2018.

[10] Edward Howorka. A characterization of distance-Hereditary graphs. *Quart. J. Math. Oxford (2)*, 28:417–420, 1977.

[11] Jessica Shi. Enumeration of unlabeled graph classes, manuscript, 2015. (available at [`https://www.cs.princeton.edu/sites/default/files/uploads/jessica_shi.pdf`], accessed on July 31, 2021.)

[12] Serafino Cicerone and Gabriele D. Stefano, "On the extension of bipartite to parity graphs," *Discrete Applied Mathematics*, vol. 95, pp. 181–195, 1999.

[13] Emeric Gioan and Christophe Paul, "Split decomposition and graph-labelled trees: Characterizations and fully dynamic algorithms for totally decomposable graphs," *Discrete Applied Mathematics*, vol. 160, no. 6, pp. 708–733, 2012.

[14] Serafino Cicerone and Gabriele D. Stefano, "Graph classes between parity and distance-hereditary graphs," *Discrete Applied Mathematics*, vol. 95, pp. 197–216, 1999.

[15] Donald E. Knuth. Sorting and Searching, volume 3 of *The Art of Computer Programming*. Addison-Wesley Publishing Company, 2nd edition, 1998.

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: *Introduction to Algorithms*, 3rd Edition. MIT Press 2009.

[17] Kazuaki Yamazaki, Toshiki Saitoh, Masashi Kiyomi, Ryuhei Uehara, Enumeration of nonisomorphic interval graphs and nonisomorphic permutation graphs. *Theoret. Comput. Sci.* 806, 323–331, 2020.

[18] Shin-ichi Nakano and Takeaki Uno. Constant time generation of trees with specified diameter. In *Graph-Theoretic Concepts in Computer Science (WG 2004)*, pages 33–45. Lecture Notes in Computer Science Vol. 3353, Springer-Verlag, 2005.

[19] Ryuhei Uehara. Graph Catalogs. 2020. URL:`http://www.jaist.ac.jp/~uehara/graphs`.