On Enumeration of Tree-Like Graphs and Pairwise Compatibility Graphs

Naveed Ahmed Azam

On Enumeration of Tree-Like Graphs and Pairwise Compatibility Graphs

Naveed Ahmed Azam

Department of Applied Mathematics and Physics
Graduate School of Informatics
Kyoto University
Kyoto, Japan

February 24, 2021

Doctoral dissertation
submitted to the
Graduate School of Informatics, Kyoto University
in partial fulfillment of
the requirement for the degree of
DOCTOR OF APPLIED MATHEMATICS

Preface

Graph enumeration with given constraints is an interesting problem considered to be one of the fundamental problems in graph theory, with many applications in natural sciences and engineering such as bioinformatics and computational chemistry. Particularly, graph enumeration helps in understanding, classification, and discovery of new chemical compounds. For an enumeration method, it is necessary to generate all possible required structures without duplication in low computational complexity, due to which designing an enumeration method is not an easy task.

This research work aims to prove some mathematical results and based on them propose new methods to enumerate tree-like graphs and pairwise compatibility graphs (PCGs) efficiently.

A polymer is a large molecule with interesting chemical properties. Polymers can be schematically represented by a graph, called polymer topology, possibly with self-loops and multi-edges, such that the graph is connected and the degree of each vertex in the graph is at least three. Classification of polymer topologies can lay a foundation for the elucidation of structural relationships between different macro-chemical molecules and their synthetic pathways. Polymer topologies are often classified based on their cycle rank, which is defined to be the number of edges that must be removed to get a simple spanning tree of the given topology. The tree-like graphs with Δ self-loops and no multiple edges contain all tree-like polymer topologies with no multiple edges and cycle rank Δ. Thus as a first step towards the enumeration of polymer topologies and multigraphs with degree constraint in general, it is interesting to count and generate all tree-like graphs with a given cycle rank.

For two integers $n \geq 1$ and $\Delta \geq 0$, we propose a method to count all non-isomorphic tree-like graphs with n vertices and cycle rank Δ. Branching algorithms and Polya's enumeration theorem are the two most commonly used methods to solve counting problems. However, branching algorithms can only count all solutions after generating each one of them, and Polya's enumeration theorem uses a group of symmetries and its cyclic index, which makes these methods inefficient or difficult to use for some problems. Unlike the traditionally used algorithms, we achieve our goal by designing an algorithm based on dynamic programming (DP) that counts the number of non-isomorphic rooted tree-like

graphs with n vertices and cycle index Δ in $\mathcal{O}(n^2(n + \Delta(n + \Delta \cdot \min\{n, \Delta\})))$ time and $\mathcal{O}(n^2(\Delta^2 + 1))$ space. By this result, we get a lower bound and an upper bound on the number of tree-like polymer topologies of chemical compounds with a given cycle rank.

To generate all non-isomorphic tree-like graphs with n vertices and cycle rank Δ, we use a canonical representation of the underlying graphs instead of traditional branching algorithms. Branching algorithms generate numerous intermediate invalid structures, and hence are inefficient. Thus we first characterize tree-like graphs in terms of their canonical representations and then generate these representations to get the required graphs. Our algorithm generates all required graphs with n vertices in $\mathcal{O}(n)$ time per graph and $\mathcal{O}(n)$ space in total, without generating invalid intermediate structures. As a consequence of our method, we can get all tree-like polymer topologies with a given cycle rank.

Pairwise compatibility graphs are used to study problems as the evolutionary relationship between a set of organisms in bioinformatics and clique problem in graph theory. A graph is a PCG if it can be represented by an edge-weighted tree whose set of leaves is the set of vertices of the graph, and there is an edge between two vertices in the graph if and only if the distance between them in the tree is within a given interval. Confirming and constructing finite-size evidence that a graph is not a PCG is a challenging task since it involves: (i) a large number of tuples called configurations that consists of the graph, a tree, a correspondence between the vertices in the graph and the leaves in the tree, and a bi-partition of all pairs of non-adjacent vertices in the graph; and (ii) an infinite search space of weights.

Motivated from the application and theoretical aspects of PCGs, we propose two linear programming (LP) formulations to construct finite-size evidence to prove if a graph is a PCG or not due to a given configuration. We prove that the feasibility of one formulation implies the infeasibility of the other formulation, and hence the solution to these formations will serve as finite-size evidence for the graph to be PCG or not. We then prove a sufficient condition for a graph to be PCG based on an integer linear programming (ILP) formulation where we try to construct a configuration and weights in a restricted domain due to which the graph is PCG. Furthermore, we characterize the configurations with four vertices such that the LP formulation is feasible in terms of a system of linear inequalities.

Finally, we propose a method to enumerate all PCGs with a given number of vertices based on the newly discovered theoretical results. To handle the difficulty of a large number of configurations, we propose a PCG generator that heuristically generates PCGs, and algorithms to generate all configurations that

do not contain a configuration with four vertices for which the LP formulation is infeasible. Our method is the first enumeration method to generate all PCGs with a given number of vertices. By using this method, we enumerated all PCGs with eight and nine vertices and proved that there are exactly seven and 1,494 minimal non-PCGs, respectively.

We believe that the work described in this thesis will help in developing the related fields in theoretical and application aspects.

Kyoto, January 2021
Naveed Ahmed Azam

Acknowledgement

This thesis would not have been possible without the help, support, and guidance of many others.

First of all, my sincere and heartfelt gratitude goes to Professor Hiroshi Nagamochi, my supervisor, for accepting me in the Discrete Mathemtaics Laboratory. During my stay as a research student and then as a Ph.D. candidate, his enthusiastic guidance, thorough discussions, and persistent encouragement allowed me to grow in many aspects. His precise comments and expert opinions have significantly improved the quality of this research work. Indeed, I feel lucky to have such a supportive and clear-minded mentor whose considerable help made it possible to complete the whole work. It will be an honor for me to keep on benefiting from his experience and expertise for the rest of my life.

I also express my sincere thanks to Professor Aleksandar Shurbevski for providing great academic support and guidance throughout my academic pursuit. His kind and welcoming attitude always helped me to freely discuss any point of my research, which eventually helped in improving the quality of this research and broaden my horizons. I hope to continue benefiting from his expertise in the following years.

I am also grateful to the other Discrete Mathematics Laboratory members for their cooperative attitude that made my transition to a new environment comfortable and pleasant.

I am immensely thankful to the Ministry of Education, Culture, Sports, Science and Technology (MEXT) and Japan Society for Promotion of Sciences (JSPS) for granting me their prestigious fellowship awards and research grants.

I am also grateful to Professor Yoshito Ohta and Professor Nobuo Yamashita for being part of my dissertation review committee. Their comments and suggestions helped me to improve the quality of exposition in various places.

I owe special gratitude to my family back in Pakistan. Certainly, their continuous support and prayers helped me in facing all challenges in my life.

Finally, I must thank my wife, Mamoona Ghafoor, for her tireless continuous support on this challenging journey. I am grateful to her for managing the home and taking care of the kids on her own. The future will bring many challenges yet, and I hope to face them with mutual support from my wife and charming kids, Minahil, Aiman, and Areeba.

Contents

List of Figures

List of Algorithms

1 Introduction

Counting and generation of discrete objects are two fundamental problems in combinatorial mathematics [9, 12, 24, 44, 51, 52, 54, 64, 70], and have many applications in applied fields such as graph theory, chemoinformatics, bioinformatics, and material informatics [4–7, 13, 30, 34, 37, 39, 40, 45, 49, 50, 53, 55, 60, 61, 69]. The counting problem asks to count all possible objects under given constraints. On the other hand, the generation problem asks to list all possible objects under given constraints. The counting or generation problem is called an enumeration problem and the methods that are used to solve these problems are called enumeration methods.

For an enumeration method, it is necessary to satisfy the following three conditions:

(i) Consider all solutions: The method does not miss any of the required objects;

(ii) Avoid duplication: The method does not count and generate isomorphic objects; and

(iiii) Low computational complexity: The method can count and generate all solutions in low time and space complexity.

Designing such a method is not an easy task, because of the underlying symmetries and the computation difficulty for their detection.

Counting and generation of chemical compounds have a long history and numerous applications in designing novel drugs [4, 13, 15, 28, 35, 39, 43, 45, 49, 68, 69] and structure elucidation [14, 26, 29, 31, 50]. The problem of counting and generation of chemical compounds can be viewed as the problem of enumerating graphs with given constraints. Several chemical compound generation methods have been proposed [30, 34, 37, 40, 55, 60, 61], where some methods [34, 55] focus on general chemical compounds, while the other methods [1, 30, 37, 60, 61] deal with restricted chemical graphs. Enumeration of restricted chemical compounds with specialized tools is more efficient than with the tools which use general graph structures [38, 59]. This led to a new trend of developing efficient enumeration of restricted chemical compounds in the field of chemoinformatics [63].

A polymer is a large molecule with interesting chemical properties consisting of many sub-molecules. From a graph-theoretic perspective, we represent the structure of a polymer with a graph G called *polymer topology*, possibly with self-loops and multi-edges, such that G is connected and the degree of each vertex in G is at least three [36]. For a chemical graph, we get its polymer topology by repeatedly

(i) removing the vertices of degree one; and

(ii) replacing each vertex u of degree two and the edges uv and uw incident to u by an edge vw, where vw is a self-loop when $v = w$.

For example, the polymer topologies of the chemical compounds remdesivir $\mathtt{C}_{27}\mathtt{H}_{35}\mathtt{N}_{6}\mathtt{O}_{8}\mathtt{P}$ (Figure 1.1(a)) and dexamethasone $\mathtt{C}_{22}\mathtt{H}_{29}\mathtt{FO}_{5}$ (Figure 1.1(c)) are illustrated in Figure 1.1(b),(d), respectively. Observe that two different chemical compounds can have the same polymer topology.

Tezuka and Oike [62] pointed out that a classification of polymer topologies will lay a foundation for the elucidation of structural relationships between different macro-chemical molecules and their synthetic pathways. Different kinds of graph-theoretic approaches have been applied to classify and enumerate polymer topologies [33, 71]. For a connected graph G, possibly with self-loops and multi-edges, the *cycle rank* is defined to be the number of edges that must be removed to get a simple spanning tree of G. Recently, Haruna et al. [36] proposed a method to enumerate all polymer topologies with cycle rank up to five.

Pairwise compatibility graphs (PCGs) is a class of graphs that is introduced as follows [42]. For a tree T, a non-negative real-valued edge weight function w on T and two non-negative reals $d_{\min}$ and $d_{\max}$, the *pairwise compatibility graph* $\mathrm{PCG}(T, w, d_{\min}, d_{\max})$ is defined to be the unweighted simple undirected graph G such that

(i) the set of vertices in G is the set of leaves in T; and

(ii) any two vertices u and v are adjacent in G if and only if the distance between the leaves u and v in T is in the closed interval $[d_{\min}, d_{\max}]$.

In such a case, we call the tree T a *witness tree* for the PCG G. Fig. 1.2 illustrates an example of a PCG with five vertices.

PCGs received great attention in the field of bioinformatics and graph theory, such as the study on the evolutionary history of a given set of organisms [42]. Kearney et al. [42] introduced PCGs to create a link between the sample problem of pairwise leaf distance for a phylogenetic tree and the clique problem. A *phylogenetic tree* is an evolutionary tree that represents some relationships between a given set of organisms. The reconstruction of phylogenetic tree is one of the fundamental problems of bioinformatics. The input of the *sampling problem of*

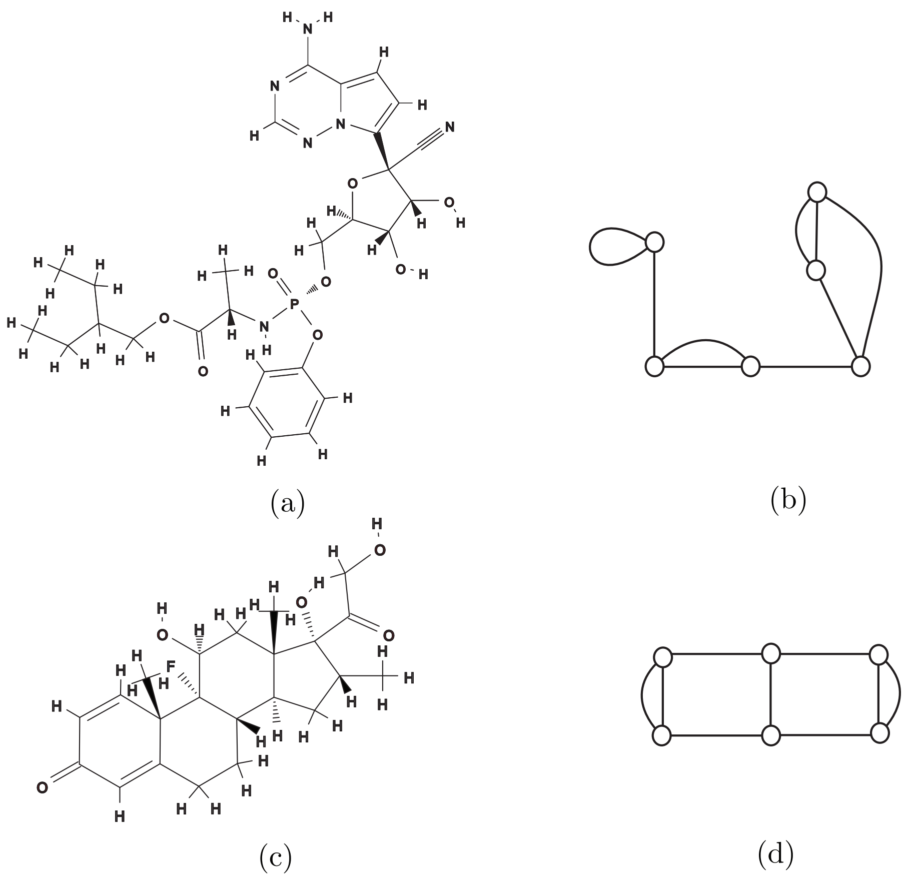


Figure 1.1. The chemical compounds remdesivir $\mathtt{C_{27}H_{35}N_6O_8P}$ and dexamethasone $\mathtt{C_{22}H_{29}FO_5}$ and their polymer topologies: (a) the chemical structure of remdesivir $\mathtt{C_{27}H_{35}N_6O_8P}$ taken from PubChem; (b) the polymer topology of remdesivir with cycle rank 4; (c) the chemical structure of dexamethasone $\mathtt{C_{22}H_{29}FO_5}$ taken from PubChem; and (d) the polymer topology of dexamethasone with cycle rank 4.

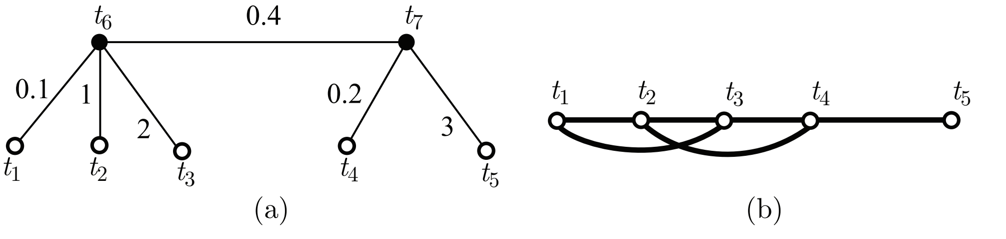


Figure 1.2. An example of a PCG: (a) A tree T with an edge weight function w; and (b) The PCG$(T, w, 1.1, 3.2)$. For example, the distance between t_1 and t_4 is $0.7 \notin [1.1, 3.2]$, and therefore there is no edge between t_1 and t_4 in the graph PCG$(T, w, 1.1, 3.2)$.

pairwise leaf distance is a phylogenetic tree, two integers i and j and two non-negative real numbers $d_{\min}$ and $d_{\max}$. The output of this problem is a set of size i consists of sample of j leaves that are selected uniformly at random from the tree such that the distance between any two leaves in the sample is in the open interval $(d_{\min}, d_{\max})$ [42]. Given a graph and an integer k, the *clique problem* asks to confirm if the graph contains a complete graph with at least k vertices. Thus Kearney et al. [42] has reduced the sampling problem of pairwise leaf distance to the clique problem for the PCG corresponding to the edge weighted phylogenetic tree, $d_{\min}$ and $d_{\max}$

Early on, Kearney et al. [42] conjectured that all graphs are PCGs, and Calamoneri et al. [20] proved that all graphs with at most seven vertices are PCGs. However, the conjecture has been refuted by Yanhaona et al. [67] by constructing a counter example of a non-PCG (NPCG) with 15 vertices. Baiocchi et al. [10] and Durocher et al. [27] constructed NPCGs with nine and eight vertices. Fig. 1.3 illustrates NPCGs with eight vertices given in [10] and [27].

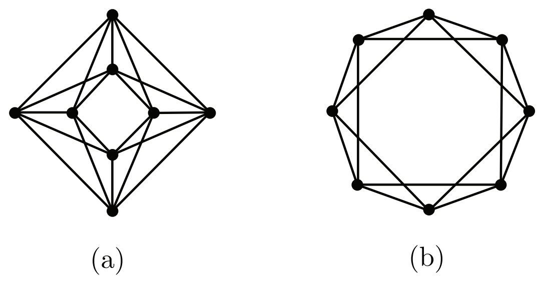


Figure 1.3. Two known NPCGs with eight vertices: (a) The NPCG given in [10]; and (b) The NPCG given in [27].

The *pairwise compatibility tree construction problem* (PCTCP) introduced by Kearney et al. [42] is one of the fundamental problems on PCGs. The input to the PCTCP is an unweighted simple undirected graph G with n vertices. The output of the PCTCP is an "evidence" (or a proof) to show if the given graph G is a PCG or not. It is important to mention that if G is a PCG, then a tuple $(T, w, d_{\min}, d_{\max})$ such that $G = \text{PCG}(T, w, d_{\min}, d_{\max})$ will be such evidence. However, when G is not a PCG, then how can we construct evidence with a finite size to verify that $G \neq \text{PCG}(T, w, d_{\min}, d_{\max})$ for any tree T with n leaves, non-negative real valued function w, and non-negative real numbers $d_{\min}$ and $d_{\max}$? Furthermore, an infinite search space of an edge weight function w and two reals $d_{\min}$ and $d_{\max}$ would be necessary to confirm that G is not a PCG. Kearney et al. [42] developed a relation between the PCTCP and the well-known clique problem: They proved that a polynomial-time algorithm for solving the

PCTCP leads to a polynomial-time algorithm for solving the clique problem in some special graph classes. It is widely believed among the research community that the PCTCP is NP-hard [18, 27]. Similarly, several variants of PCGs have been studied (see [11, 16, 17, 19, 21, 48, 58, 66]).

Motivated from the application and theoretical aspects of the polymer topologies and PCGs, we prove some mathematical results which are then used to propose enumeration methods for these graph classes. Our contributions are: (i) Propose an efficient method to count all tree-like graphs with a given cycle rank in Chapter 2 as a first step towards enumeration of polymer topologies and graphs with degree constraint in general; (ii) Characterize tree-like graphs with a given cycle rank in terms of canonical representation based on which we propose an algorithm to generate such graphs in Chapter 3; (iii) Derive some theoretical results related to PCGs to efficiently solve the PCTCP in Chapter 4; and (iv) By using the results proved in (iii), propose a method to enumerate all PCGs with a given number of vertices in Chapter 5.

2 Counting Tree-Like Graphs with a Given Cycle Rank

2.1 Introduction

Different kinds of methods are used to solve counting problem, where branching algorithms and Polya's enumeration theorem are the two most commonly used methods for this problem. In branching algorithms, the computation is performed by following a computation tree, and the required solutions are attained at the leaves of the computation tree. It is important to mention that the branching algorithms can only count all solutions after generating each one of them, and therefore they are inefficient for the problem where we first want to know the size of the solution space before the generation of solutions.

The well-known Polya's enumeration theorem [56, 57] is used for counting all distinct objects. The idea of this method is to use the cyclic index of the group of symmetries of the underlying object to develop a generating function, which is then used to count all possible objects. Note that finding the group of symmetries and its cyclic index is a challenging task, which may make the use of Polya's theorem harder for some problems.

The drawback of branching algorithms discussed above and the difficulty of using Polya's theorem necessitate the exploration of new enumeration methods to solve counting problems efficiently.

Dynamic programming (DP) is an other algorithm paradigm where

(i) the original problem is partitioned into subproblems that satisfy some recursive relations; and

(ii) the union of the solution sets of the subproblems is equal to the solution set of the original problem.

Unlike branching algorithms and Polya's theorem, the main advantage of using the DP is that we can count all non-isomorphic structures without their generation and calculation of their group of symmetries.

For a multi-graph G, we define the *skeleton* to be the simple graph obtained by removing all self-loops and multiple edges from G. Notice that the class of graphs with a tree skeleton, $\Delta \geq 0$ self-loops, and no multiple edges contains all tree-like polymer topologies with cycle rank Δ, and therefore, it is an interesting problem to count and generate all such graphs as a first step towards the enumeration of polymer topologies and degree bounded graphs in general. Figure 2.1 illustrates examples of chemical compounds that have tree-like polymer topologies with self-loops and no multiple edges.

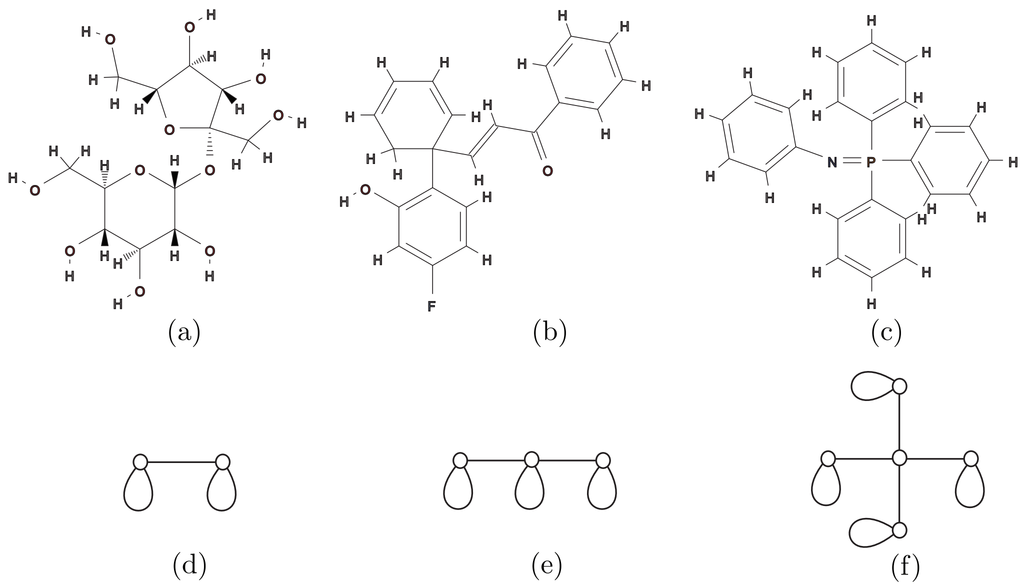


Figure 2.1. Three chemical compounds with their tree-like polymer topologies containing self-loops: (a–c) the chemical structures of $C_{12}H_{22}O_{11}$, $C_{21}H_{17}FO_2$, and $C_{24}H_{20}NP$ with CIDs 5988, 137,321,354, 75,352, respectively, obtained from the PubChem database; (d–f) the polymer topologies of the chemical structures in (a–c) with cycle ranks 2–4, respectively.

We use DP to count all mutually non-isomorphic graphs with n vertices, Δ self-loops, no multi-edges and tree skeleton. To achieve this goal, we count the number of non-isomorphic rooted graphs with tree skeleton, n vertices, Δ self-loops and no multi-edges, since every tree can be uniquely viewed as a rooted tree by either regarding its unicentroid as the root, or in the case of bicentroid, by introducing a virtual vertex on the bicentroid and assuming the virtual vertex to be the root. As an application of our results, we get lower and upper bounds on the number of tree-like polymer topologies with self-loops of a given cycle rank.

The rest of the chapter is organized as follows: Section 2.2 reviews some notions and results related to graph theory. Section 2.3 explains our graph counting method. In Section 2.4, we discuss some experimental results and application of our counting method. Section 2.5 makes some concluding remarks.

2.2 Preliminaries

Throughout Chapter 2 and 3, the term *graph* stands for an undirected graph with no multi-edges and possibly with self-loops unless stated otherwise. Let G be a graph. We denote an edge between two vertices u and v in G by uv where we consider $uv = vu$. Let $V(G)$ and $E(G)$ denote the vertex set and edge set of G, respectively. Let $\mathrm{n}(G)$ denote $|V(G)|$ and $s(G)$ denote the number of self-loops in G. For a vertex $v \in V(G)$, we denote by $s(v)$ the number of self-loops on the vertex v. We define size $\mathrm{s}(G)$ of graph G to be the sequence $(\mathrm{n}(G), s(G))$. For a vertex v in G, let $N_G(v)$ denote the set of vertices incident to v except v itself and the *degree* $\deg_G(v)$ of v in G is defined to be $|N_G(v)|$. A graph H with the properties $V(H) \subseteq V(G)$ and $E(G) \subseteq E(G)$ is called a *subgraph* of G. A *simple path* between two distinct vertices $u, v \in V(G)$ is defined to be a subgraph P of G with vertex set $V(P) = \{u = w_1, w_2, \ldots, v = w_k\}$ and edge set $E(P) = \{w_i w_{i+1} \mid 1 \leq i \leq k-1\}$. A graph is called a *connected graph* if there is a path between any two distinct vertices in the graph. A *connected component* of a graph G is defined to be a maximal connected subgraph H of G, i.e., for any vertex $v \in V(G) \setminus V(H)$ it holds that every subgraph with the vertex set $V(H) \cup \{v\}$ is disconnected.

For a graph G, we define the *skeleton* $\gamma(G)$ of G to be the simple graph obtained by removing all self-loops from G. A graph with a fixed vertex r is called a *rooted graph* with root r. For a rooted graph G, we denote by r_G the root of G. For a rooted graph G with root r_G, we define the *rooted skeleton* $\gamma(G)$ of G to be the rooted simple graph obtained by removing all self-loops from G with root r_G.

Two rooted graphs G and G' are called *isomorphic* if there exists a bijection $\sigma : V(G) \to V(G')$ such that

(i) $\sigma(r_G) = r'_G$;

(ii) for each vertex $v \in V(G)$, it holds that $s(v) = s(\sigma(v))$; and

(iii) for any two vertices $u, v \in V(G)$, it holds that $uv \in E(G')$ if and only if $\sigma(u)\sigma(v) \in E(G')$.

By Jordan [41], any simple tree with $n \geq 1$ vertices has either a unique vertex or edge, the removal of which creates connected components with at most $\lfloor (n-1)/2 \rfloor$ or exactly $n/2$ vertices, respectively. Such a vertex is called the *unicentroid*, the edge is called the *bicentroid*, and collectively they are called the *centroid* of the tree. It is important to note that there exits a bicentroid only for trees with an even number of vertices. Note that any tree can be uniquely viewed as a rooted tree by either regarding its unicentroid as the root, or in the case of a bicentroid, by introducing a virtual vertex on the bicentroid and assuming the virtual vertex as the root.

Let T be a rooted tree. Let r_T denote the root of T. For any two distinct vertices $u, v \in V(T)$, let $P_T(u, v)$ denote the unique simple path between them in T. For a vertex $v \in V(T) \setminus \{r_T\}$, we define the *ancestors* of v to be the vertices on the path $P_T(v, r_T)$ other than v. If u is an ancestor of v, then we call v a *descendant* of u. For a vertex $v \in V(T) \setminus \{r_T\}$, the *parent* $p(v)$ of v is defined to be the ancestor u of v such that $u \in N_T(v)$. We call the vertex v a *child* of $p(v)$. Two vertices with the same parent in T are called *siblings*.

Let $n \geq 1$ and $\Delta \geq 0$ be two integers. We denote by $\mathcal{H}(n, \Delta)$ a maximal set of mutually non-isomorphic rooted graphs with a tree skeleton, n vertices, and Δ self-loops. We denote $|\mathcal{H}(n, \Delta)|$ by $h(n, \Delta)$. Recall that the cycle rank of each graph in $\mathcal{H}(n, \Delta)$ is Δ. Let H be a rooted graph in $\mathcal{H}(n, \Delta)$. For a vertex $v \in V(H)$, let H_v denote the subgraph of H rooted at v induced by v and its descendants in the rooted skeleton $\gamma(H)$. For a vertex $v \in N_H(r_H)$ of root r_H of H, we call the subgraph H_v a *root-subgraph* of H.

2.3 Counting Tree-Like Graphs with a Given Number of Vertices and Self-loops

We develop a method to compute for any two integers $n \geq 1$ and $\Delta \geq 0$, the size $h(n, \Delta)$ of a maximal set $\mathcal{H}(n, \Delta)$ of mutually non-isomorphic rooted graphs with n vertices and Δ self-loops; i.e., we are interested in the following problem.

Counting Problem
Input: Two integers $n \geq 1$ and $\Delta \geq 0$.
Output: $h(n, \Delta)$.

We solve this problem by using dynamic programming based on the information of the number of vertices and self-loops in the subgraphs rooted at the children of the root of each graph in $\mathcal{H}(n, \Delta)$. We define the following notions.

Let $n \geq 1$ and $\Delta \geq 0$ be any two integers. For each graph $H \in \mathcal{H}(n, \Delta)$, we define

$$\begin{aligned}
\mathrm{Max_v}(H) &\triangleq \max(\{|V(H_v)| \mid v \in N_H(r_H)\} \cup \{0\}), \\
\mathrm{Max_s}(H) &\triangleq \max(\{s(H_v) \mid v \in N_H(r_H), |V(H_v)| = \mathrm{Max_v}(H)\} \cup \{0\}).
\end{aligned}$$

Note that for any graph $H \in \mathcal{H}(1,\Delta)$, it holds that $\mathrm{Max}_{\mathrm{v}}(H) = 0$ and $\mathrm{Max}_{\mathrm{s}}(H) = 0$.

Let $m, d \geq 0$ be any two integers. We define

$$\mathcal{H}(n,\Delta,m_{\leq},d_{\leq}) \triangleq \{H \in \mathcal{H}(n,\Delta) \mid \mathrm{Max}_{\mathrm{v}}(H) \leq m, \mathrm{Max}_{\mathrm{s}}(H) \leq d\}.$$

Observe that by the definition of $\mathcal{H}(n,\Delta,m_{\leq},d_{\leq})$ it holds that

(i) $\mathcal{H}(n,\Delta,m_{\leq},d_{\leq}) = \mathcal{H}(n,\Delta,n-1_{\leq},d_{\leq})$ if $m \geq n$;

(ii) $\mathcal{H}(n,\Delta,m_{\leq},d_{\leq}) = \mathcal{H}(n,\Delta,m_{\leq},\Delta_{\leq})$ if $d \geq \Delta+1$; and

(iii) $\mathcal{H}(n,\Delta) = \mathcal{H}(n,\Delta,n-1_{\leq},\Delta_{\leq})$.

Therefore, from now on, we assume that $m \leq n-1$ and $d \leq \Delta$. Further, by the definition of $\mathcal{H}(n,\Delta,m_{\leq},d_{\leq})$ it holds that $\mathcal{H}(n,\Delta,m_{\leq},d_{\leq}) \neq \emptyset$ (resp., $\mathcal{H}(n,\Delta,m_{\leq},d_{\leq}) = \emptyset$) if "$n = 1$" or "$n-1 \geq m \geq 1$" (resp., otherwise ($n \geq 2$ and $m = 0$)).

We define

$$\mathcal{H}(n,\Delta,m_{=},d_{\leq}) \triangleq \{H \in \mathcal{H}(n,\Delta,m_{\leq},d_{\leq}) \mid \mathrm{Max}_{\mathrm{v}}(H) = m\}.$$

It follows from the definition of $\mathcal{H}(n,\Delta,m_{=},d_{\leq})$ that $\mathcal{H}(n,\Delta,m_{=},d_{\leq}) \neq \emptyset$ (resp., $\mathcal{H}(n,\Delta,m_{=},d_{\leq}) = \emptyset$) if "$n = 1$" or "$n-1 \geq m \geq 1$" (resp., otherwise ($n \geq 2$ and $m = 0$)). Further we have the following relations.

$$\mathcal{H}(n,\Delta,m_{\leq},d_{\leq}) = \mathcal{H}(n,\Delta,0_{=},d_{\leq}) \text{ if } m = 0, \tag{2.3.1}$$

$$\mathcal{H}(n,\Delta,m_{\leq},d_{\leq}) = \mathcal{H}(n,\Delta,m-1_{\leq},d_{\leq}) \cup \mathcal{H}(n,\Delta,m_{=},d_{\leq}) \text{ if } m \geq 1, \tag{2.3.2}$$

where $\mathcal{H}(n,\Delta,m-1_{\leq},d_{\leq}) \cap \mathcal{H}(n,\Delta,m_{=},d_{\leq}) = \emptyset$ for $m \geq 1$.

Next we define

$$\mathcal{H}(n,\Delta,m_{=},d_{=}) \triangleq \{H \in \mathcal{H}(n,\Delta,m_{=},d_{\leq}) \mid \mathrm{Max}_{\mathrm{s}}(H) = d\}.$$

Note that if "$n = 1$ and $d = 0$" or "$n-1 \geq m \geq 1$" (resp., otherwise ("$n = 1$ and $d \geq 1$" or "$n \geq 2$ and $m = 0$")), then by the definition of $\mathcal{H}(n,\Delta,m_{=},d_{=})$ it holds that $\mathcal{H}(n,\Delta,m_{=},d_{=}) \neq \emptyset$ (resp., $\mathcal{H}(n,\Delta,m_{=},d_{=}) = \emptyset$). Furthermore, we get the following relations for $\mathcal{H}(n,\Delta,m_{=},d_{\leq})$.

$$\mathcal{H}(n,\Delta,m_{=},d_{\leq}) = \mathcal{H}(n,\Delta,m_{=},0_{=}) \text{ if } d = 0, \tag{2.3.3}$$

$$\mathcal{H}(n,\Delta,m_{=},d_{\leq}) = \mathcal{H}(n,\Delta,m_{=},d-1_{\leq}) \cup \mathcal{H}(n,\Delta,m_{=},d_{=}) \text{ if } d \geq 1, \tag{2.3.4}$$

where $\mathcal{H}(n,\Delta,m_{=},d-1_{\leq}) \cap \mathcal{H}(n,\Delta,m_{=},d_{=}) = \emptyset$ for $d \geq 1$.

Let $n-1 \geq m \geq 0$, and $\Delta \geq d \geq 0$ be four integers. Let $h(n,\Delta,m_{\leq},d_{\leq})$, $h(n,\Delta,m_{=},d_{\leq})$ and $h(n,\Delta,m_{=},d_{=})$ denote the number of elements in the families $\mathcal{H}(n,\Delta,m_{\leq},d_{\leq})$, $\mathcal{H}(n,\Delta,m_{=},d_{\leq})$ and $\mathcal{H}(n,\Delta,m_{=},d_{=})$, respectively. We discuss recursive relations for $h(n,\Delta,m_{\leq},d_{\leq})$ and $h(n,\Delta,m_{=},d_{\leq})$ in Lemma 2.1.

Lemma 2.1. *For any four integers $n-1 \geq m \geq 0$, and $\Delta \geq d \geq 0$, it holds that*

(i) $h(n, \Delta, m_{\leq}, d_{\leq}) = h(n, \Delta, 0_{=}, d_{\leq})$ *if* $m = 0$*;*

(ii) $h(n, \Delta, m_{\leq}, d_{\leq}) = h(n, \Delta, m-1_{\leq}, d_{\leq}) + h(n, \Delta, m_{=}, d_{\leq})$ *if* $m \geq 1$*;*

(iii) $h(n, \Delta, m_{=}, d_{\leq}) = h(n, \Delta, m_{=}, 0_{=})$ *if* $d = 0$*; and*

(iv) $h(n, \Delta, m_{=}, d_{\leq}) = h(n, \Delta, m_{=}, d-1_{\leq}) + h(n, \Delta, m_{=}, d_{=})$ *if* $d \geq 1$*.*

Proof. The case (i) follows by Equation (2.3.1). The case (ii) follows by Equation (2.3.2) and the fact that for $m \geq 1$ it holds that $\mathcal{H}(n, \Delta, m-1_{\leq}, d_{\leq}) \cap \mathcal{H}(n, \Delta, m_{=}, d_{\leq}) = \emptyset$. By Equation (2.3.3) the case (iii) follows. The case (iv) follows by Equation (2.3.4) and the fact that for $d \geq 1$ it holds that $\mathcal{H}(n, \Delta, m_{=}, d-1_{\leq}) \cap \mathcal{H}(n, \Delta, m_{=}, d_{=}) = \emptyset$. □

Next we discuss some boundary conditions for our DP to compute $h(n, \Delta)$.

Lemma 2.2. *For any four integers $n-1 \geq m \geq 0$, and $\Delta \geq d \geq 0$, it holds that*

(i) $h(n, \Delta, 0_{=}, d_{=}) = 1$ *(resp.,* $h(n, \Delta, 0_{=}, d_{=}) = 0$*) if* $n = 1$ *and* $d = 0$ *(resp., otherwise ("*$n = 1$ *and* $d \geq 1$*" or "*$n \geq 2$*"));*

(ii) $h(n, \Delta, 0_{=}, d_{\leq}) = h(n, \Delta, 0_{\leq}, d_{\leq}) = 1$ *(resp.,* $h(n, \Delta, 0_{=}, d_{\leq}) = h(n, \Delta, 0_{\leq}, d_{\leq}) = 0$*) if* $n = 1$ *(resp., otherwise (*$n \geq 2$*));*

(iii) $h(n, \Delta, 1_{=}, d_{=}) = 1$ *if "*$n = 2$*" or "*$n \geq 3$ *and* $d = 0$*"; and*

(iv) $h(n, \Delta, 1_{=}, d_{\leq}) = h(n, \Delta, 1_{\leq}, d_{\leq}) = d + 1$ *if "*$n = 2$*" or "*$n \geq 3$ *and* $d = 0$*".*

Proof. (i) The result follows from the definition of $\mathcal{H}(n, \Delta, 0_{=}, d_{=})$, since a graph H with $\max_{\mathrm{v}}(H) = 0$ exists if and only if $|V(H)| = 1$ and $\max_{\mathrm{s}}(H) = 0$.

(ii) By Lemma 2.1(i), (ii) and (iv) it holds that $h(n, \Delta, 0_{\leq}, d_{\leq}) = h(n, \Delta, 0_{=}, d_{\leq}) = \sum\limits_{p=0}^{d} h(n, \Delta, 0_{=}, p_{=})$. This and Lemma 2.2(i) imply the required result.

(iii) When $n \geq 2$, then for any graph $H \in \mathcal{H}(n, \Delta, 1_{=}, d_{=})$ it holds that $|N_H(r_H)| = n - 1$. Thus for each $v \in N_H(r_H)$ it holds that $|V(H_v)| = 1$ and $s(H_v) = d$ if "$n = 2$" or "$n \geq 3$ and $d = 0$", i.e., $H_v \in \mathcal{H}(1, d, 0_{\leq}, d_{\leq})$. But by Lemma 2.2(ii) it holds that $h(1, d, 0_{\leq}, d_{\leq}) = 1$. Hence we have the required result.

(iv) Let "$n = 2$" or "$n \geq 3$ and $d = 0$". By Lemma 2.1(iii) and (iv) it holds that $h(n, \Delta, 1_{=}, d_{\leq}) = \sum\limits_{p=0}^{d} h(n, \Delta, 1_{=}, p_{=})$. This and Lemma 2.2(iii) imply that

$$h(n, \Delta, 1_{=}, d_{\leq}) = d + 1. \tag{2.3.5}$$

Furthermore, by Lemma 2.1(iii) it holds that $h(n, \Delta, 1_{\leq}, d_{\leq}) = h(n, \Delta, 0_{=}, d_{\leq}) + h(n, \Delta, 1_{=}, d_{\leq})$. By Lemma 2.2(ii), we have $h(n, \Delta, 1_{\leq}, d_{\leq}) = h(n, \Delta, 1_{=}, d_{\leq})$. Hence the result follows by Equation (2.3.5).

□

By Lemma 2.2, we can get that $h(1, \Delta) = 1$ and $h(2, \Delta) = \Delta + 1$. Furthermore, Lemma 2.1(i)–(iv) give recursive relations for $h(n, \Delta, m_{\leq}, d_{\leq})$ and $h(n, \Delta, m_{=}, d_{\leq})$ which depend on $h(n, \Delta, m_{=}, d_{=})$. Thus for $n \geq 3$, $m \geq 1$, and $\Delta \geq d \geq 0$, our next goal is to develop a recursive relation for $h(n, \Delta, m_{=}, d_{=})$. For any graph $H \in \mathcal{H}(n, \Delta, m_{=}, d_{=})$ and any vertex $v \in N_H(r_H)$, the subgraph H_v of H satisfies exactly one of the following three conditions:

(C-1) $|V(H_v)| = m$ and $s(H_v) = d$.

(C-2) $|V(H_v)| = m$ and $0 \leq s(H_v) < d$.

(C-3) $|V(H_v)| < m$ and $0 \leq s(H_v) \leq \Delta$.

For any graph $H \in \mathcal{H}(n, \Delta, m_{=}, d_{=})$, we define the *residual graph* of H to be the subgraph of H rooted at r_H induced by the vertices $V(H) \setminus \bigcup_{\substack{v \in N_H(r_H), \\ H_v \in \mathcal{H}(m, d, m-1_{\leq}, d_{\leq})}} V(H_v)$.

Note that the residual graph of a graph H has at least one vertex, i.e., the root of H. We give an illustration of a residual graph in Figure 2.2.

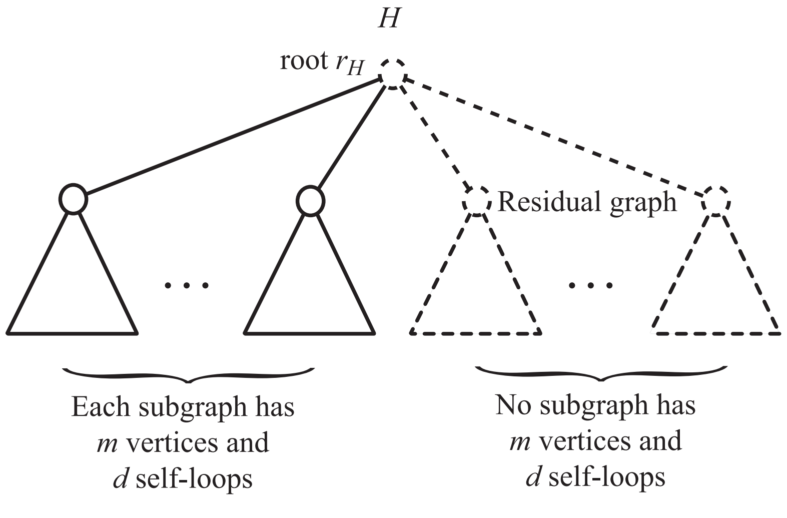

H
root r_H
Residual graph
Each subgraph has m vertices and d self-loops
No subgraph has m vertices and d self-loops

Figure 2.2. An illustration of a residual graph, where $H \in \mathcal{H}(n, \Delta, m_{=}, d_{=})$ and the residual graph of H is shown by dashed lines.

Lemma 2.3. *For any four integers $n \geq 3$, $m \geq 1$, and $\Delta \geq d \geq 0$, and a graph $H \in \mathcal{H}(n, \Delta, m_{=}, d_{=})$, let $q = |\{v \in N_H(r_H) \mid H_v \in \mathcal{H}(m, d, m - 1_{\leq}, d_{\leq})\}|$. Then it holds that*

(i) *$1 \le q \le \lfloor (n-1)/m \rfloor$ with $q \le \lfloor \Delta/d \rfloor$ when $d \ge 1$.*

(ii) *The residual graph of H belongs to exactly one of the families $\mathcal{H}(n-qm, \Delta - dq, m_=, \min\{\Delta - dq,\ d-1\}_\le)$ and $\mathcal{H}(n-qm, \Delta - dq, \min\{n-qm-1, m-1\}_\le, \Delta - dq_\le)$.*

Proof. (i) Since $H \in \mathcal{H}(n, \Delta, m_=, d_=)$, there exists at least one vertex $v \in N_H(r_H)$ such that $H_v \in \mathcal{H}(m, d, m-1_\le, d_\le)$. This implies that $q \ge 1$. Also, it holds that $n-1 \ge mq$ and $\Delta \ge dq$. This implies that $q \le \lfloor (n-1)/m \rfloor$ with $q \le \lfloor \Delta/d \rfloor$ when $d \ge 1$.

(ii) Let K denote the residual graph of H. By the definition of K it holds that $K \in \mathcal{H}(n-mq, \Delta-dq, n-mq-1_\le, \Delta-dq_\le)$. Furthermore, for each vertex $v \in N_H(r_H) \cap V(K)$, the graph H_v satisfies exactly one of the conditions (C-2) and (C-3). Now, if there exists a vertex $v \in N_H(r_H) \cap V(K)$ such that H_v satisfies condition (C-2), then $d-1 \ge 0$, and hence $K \in \mathcal{H}(n-qm, \Delta - dq, m_=, \min\{\Delta - dq, d-1\}_\le)$. On the other hand, if condition (C-2) does not hold for any $v \in N_H(r_H) \cap V(K)$; i.e., either $N_H(r_H) \cap V(K) = \emptyset$ or for each $v \in N_H(r_H) \cap V(K)$ it holds that $|V(H_v)| \le \min\{n-qm-1, m-1\}$ and $0 \le s(H_v) \le \Delta - dq$, then by the definition of K it holds that $K \in \mathcal{H}(n-qm, \Delta - dq, \min\{n-qm-1, m-1\}_\le, \Delta - dq_\le)$. This completes the proof.

□

For any five integers $n \ge 3$, $m \ge 1$, $\Delta \ge d \ge 0$, and $t \ge 0$, let $c(m, d; t) \triangleq \binom{h(m,d,m-1_\le,d_\le)+t-1}{t}$ denote the number of combinations with repetition of t graphs from the family $\mathcal{H}(m, d, m-1_\le, d_\le)$. In Lemma 2.4, we give a recursive relation for $h(n, \Delta, m_=, d_=)$.

Lemma 2.4. *For any five integers $n \ge 3$, $m \ge 1$, $\Delta \ge d \ge 0$, and q, such that $1 \le q \le \lfloor (n-1)/m \rfloor$ with $q \le \lfloor \Delta/d \rfloor$ when $d \ge 1$, it holds that*

(i) *$h(n, \Delta, m_=, d_=) = \sum_q c(m, d; q) h(n-qm, \Delta, \min\{n-qm-1, m-1\}_\le, \Delta_\le)$ if $d = 0$;*

(ii) *$h(n, \Delta, m_=, d_=) = \sum_q c(m, d; q)(h(n-qm, \Delta-dq, m_=, \min\{\Delta-dq, d-1\}_\le) + h(n-qm, \Delta-dq, \min\{n-qm-1, m-1\}_\le, \Delta-dq_\le))$ if $d \ge 1$;*

(iii) *$h(n, \Delta, m_=, d_=) = \sum_q c(m, d; q-1)((h(m, d, m-1_\le, d_\le) + q-1)/q) h(n-qm, \Delta, \min\{n-qm-1, m-1\}_\le, \Delta_\le)$ if $d = 0$; and*

(iv) *$h(n, \Delta, m_=, d_=) = \sum_q c(m, d; q-1)((h(m, d, m-1_\le, d_\le) + q-1)/q)(h(n-qm, \Delta-dq, m_=, \min\{\Delta-dq, d-1\}_\le) + h(n-qm, \Delta-dq, \min\{n-qm-1, m-1\}_\le, \Delta-dq_\le))$ if $d \ge 1$.*

Proof. Let H be a graph in the family $\mathcal{H}(n, \Delta, m_=, d_=)$. By Lemma 2.3(i), there exists a unique integer q, $1 \leq q \leq \lfloor (n-1)/m \rfloor$ with $q \leq \lfloor \Delta/d \rfloor$ when $d \geq 1$, such that there are exactly q subgraphs H_v with $v \in N_H(r_H)$ and $H_v \in \mathcal{H}(m, d, m-1_{\leq}, d_{\leq})$. Further, by Lemma 2.3(ii) the residual graph of H belongs to the family $\mathcal{H}(n-qm, \Delta, \min\{n-qm-1, m-1\}_{\leq}, \Delta_{\leq})$ (resp., $\mathcal{H}(n-qm, \Delta-dq, m_=, \min\{\Delta-dq, d-1\}_{\leq}) \cup \mathcal{H}(n-qm, \Delta-dq, \min\{n-qm-1, m-1\}_{\leq}, \Delta-dq_{\leq})$) if $d = 0$ (resp., otherwise). Note that $\mathcal{H}(n-qm, \Delta-dq, m_=, \min\{\Delta-dq, d-1\}_{\leq}) \cap \mathcal{H}(n-qm, \Delta-dq, \min\{n-qm-1, m-1\}_{\leq}, \Delta-dq_{\leq}) = \emptyset$. This implies that for a fixed integer q in the range given in the lemma, the number of graphs K in the family $\mathcal{H}(n, \Delta, m_=, d_=)$ with exactly q subgraphs $K_v \in \mathcal{H}(m, d, m-1_{\leq}, d_{\leq})$, for $v \in N_K(r_K)$, are

(a) $c(m, d; q)h(n-qm, \Delta, \min\{n-qm-1, m-1\}_{\leq}, \Delta_{\leq})$ if $d = 0$; and

(b) $c(m, d; q)(h(n-qm, \Delta-dq, m_=, \min\{\Delta-dq, d-1\}_{\leq}) + h(n-qm, \Delta-dq, \min\{n-qm-1, m-1\}_{\leq}, \Delta-dq_{\leq}))$ if $d \geq 1$.

Note that, for $m = 1$ and $d = 0$, we have $1 \leq q \leq n-1$, and by Lemma 2.2(ii) it holds that $h(n-q, \Delta, 0_{\leq}, \Delta_{\leq}) = 0$ (resp., $h(n-q, \Delta, 0_{\leq}, \Delta_{\leq}) = 1$), if $1 \leq q \leq n-2$ (resp., otherwise (if $q = n-1$)). This implies that any graph $H \in \mathcal{H}(n, \Delta, 1_=, 0_=)$ has exactly $q = n-1$ subgraphs $H_v \in \mathcal{H}(1, 0, 0_{\leq}, 0_{\leq})$, for $v \in N_H(r_H)$. However, observe that for each integer $m \geq 2$ or $d \geq 1$, and q satisfying the conditions given in the lemma, there exists at least one graph $H \in \mathcal{H}(n, \Delta, m_=, d_=)$ such that H has exactly q subgraphs $H_v \in \mathcal{H}(m, d, m-1_{\leq}, d_{\leq})$, for $v \in N_H(r_H)$. Hence, this and case (a) (resp., case (b)) imply Lemma 2.4(i) (resp., Lemma 2.4(ii)).

Furthermore, it holds that

$$\begin{aligned}
c(m, d; q) &= \frac{(h(m, d, m-1_{\leq}, d) + q - 1)!}{(h(m, d, m-1_{\leq}, d) - 1)!q!} \\
&= \frac{(h(m, d, m-1_{\leq}, d) + q - 2)!}{(h(m, d, m-1_{\leq}, d) - 1)!(q-1)!} \times \frac{(h(m, d, m-1_{\leq}, d) + q - 1)}{q} \\
&= c(m, d; q-1) \times \frac{(h(m, d, m-1_{\leq}, d_{\leq}) + q - 1)}{q}.
\end{aligned}$$

Hence, Lemma 2.4(iii) and (iv) follow from Lemma 2.4(i) and (ii), respectively. □

Lemma 2.5. *For any four integers $n-1 \geq m \geq 0$, and $\Delta \geq d \geq 0$, $h(n, \Delta, m_{\leq}, d_{\leq})$ can be obtained in $\mathcal{O}(nm(n+\Delta(n+d\cdot\min\{n, \Delta\})))$ time and $\mathcal{O}(nm(\Delta(d+1)+1))$ space.*

The proof of Lemma 2.5 follows from Algorithm 1 and Lemma 2.6.

Corollary 2.1. *For any two integers $n \geq 1$ and $\Delta \geq 0$, $h(n, \Delta, n-1_{\leq}, \Delta_{\leq})$ can be obtained in $\mathcal{O}(n^2(n + \Delta(n + \Delta \cdot \min\{n, \Delta\})))$ time and $\mathcal{O}(n^2(\Delta^2 + 1))$ space.*

We design a DP algorithm to compute $h(n, \Delta)$ based on the recursive structures of $h(n, \Delta, m_{\leq}, d_{\leq})$, $h(n, \Delta, m_{=}, d_{\leq})$ and $h(n, \Delta, m_{=}, d_{=})$, $0 \leq m \leq n-1$ and $0 \leq d \leq \Delta$, as given in Lemmas 2.1 and 2.4, where $h(n, \Delta) = h(n, \Delta, n-1_{\leq}, \Delta_{\leq})$ for $n \geq 1$ and $\Delta \geq 0$. For any four integers $n-1 \geq m \geq 0$, and $\Delta \geq d \geq 0$, we present Algorithm 1 for solving the problem of calculating $h(n, \Delta, m_{\leq}, d_{\leq})$. In this algorithm, for each integers $1 \leq i \leq n, 0 \leq j \leq \Delta, 0 \leq k \leq \min\{i, m\}$, and $0 \leq p \leq \min\{j, d\}$, the variables $h\,[i, j, k_{\leq}, p_{\leq}]$, $h\,[i, j, k_{=}, p_{\leq}]$, and $h\,[i, j, k_{=}, p_{=}]$ store the values of $h(i, j, k_{\leq}, p_{\leq}), h(i, j, k_{=}, p_{\leq})$, and $h(i, j, k_{=}, p_{=})$, respectively.

Algorithm 1 DP Based Counting Algorithm for $h(n, \Delta, m_{\leq}, d_{\leq})$

```
Input: Integers $n-1 \geq m \geq 0$ and $\Delta \geq d \geq 0$.
Output: $h(n, \Delta, m_{\leq}, d_{\leq})$.
1: $h[1, j, 0_{=}, 0_{=}] := h[1, j, 0_{=}, p_{\leq}] := h[1, j, 0_{\leq}, p_{\leq}] := 1$;
   $h[i, j, 0_{=}, p_{\leq}] := h[i, j, 0_{\leq}, p_{\leq}] := 0$;
   $h[2, j, 1_{=}, p_{=}] := 1$; $h[2, j, 1_{=}, p_{\leq}] := h[2, j, 1_{\leq}, p_{\leq}] := p + 1$
   for each $2 \leq i \leq n, 0 \leq j \leq \Delta$, $0 \leq p \leq \min\{j, d\}$;
2: for $i := 3, 4, \ldots, n$ do
3:  for $j := 0, 1, \ldots, \Delta$ do
4:   for $k := 1, 2, \ldots, \min\{i, m\}$ do
5:    for $p := 0, 1, \ldots, \min\{j, d\}$ do
6:     if $p = 0$ and $k = 1$ then
7:      $h[i, j, 1_{=}, 0_{=}] := h[i, j, 1_{=}, 0_{\leq}] := h[i, j, 1_{\leq}, 0_{\leq}] := 1$
8:     else /* $p \geq 1$ or $k \geq 2$ */
9:      $c := 1$; $h[i, j, k_{=}, p_{=}] := 0$; /* Initialization */
10:      if $p = 0$ then
11:       $\ell := \lfloor (i-1)/k \rfloor$
12:      else /* $p \geq 1$ */
13:       $\ell := \min\{\lfloor (i-1)/k \rfloor, \lfloor j/p \rfloor\}$
14:      end if;
15:      for $q := 1, 2, \ldots, \ell$ do
16:       $c := c \cdot (h[k, p, k-1_{\leq}, p_{\leq}] + q - 1)/q$;
17:       if $p = 0$ then
18:        $h[i, j, k_{=}, p_{=}] := h[i, j, k_{=}, p_{=}] + c \cdot h[i-qk, j, \min\{i-kq-1, k-1\}_{\leq}, j_{\leq}]$
19:       else /* $p \geq 1$ */
20:        $h[i, j, k_{=}, p_{=}] := h[i, j, k_{=}, p_{=}] + c \cdot h[i-kq, j-pq, k_{=}, \min\{j-pq, p-1\}_{\leq}] +$
           $h[i-kq, j-pq, \min\{i-kq-1, k-1\}_{\leq}, j-pq_{\leq}]$
```

```
21:         end if
22:       end for;
23:       if $p = 0$ then /* $k \geq 2$ */
24:         $h[i, j, k_=, 0_\leq] := h[i, j, k_=, 0_=]$
25:       else /* $p \geq 1$ */
26:         $h[i, j, k_=, p_\leq] := h[i, j, k_=, p - 1_\leq] + h[i, j, k_=, p_=]$
27:       end if;
28:       $h[i, j, k_\leq, p_\leq] := h[i, j, k - 1_\leq, p_\leq] + h[i, j, k_=, p_\leq]$
29:     end if
30:    end for
31:   end for
32:  end for
33: end for;
34: Output $h[n, \Delta, m_\leq, d_\leq]$ as $h(n, \Delta, m_\leq, d_\leq)$.
```

Lemma 2.6. *For any four integers $n - 1 \geq m \geq 0$, and $\Delta \geq d \geq 0$, Algorithm 1 outputs $h(n, \Delta, m_\leq, d_\leq)$ in $\mathcal{O}(nm(n+\Delta(n+d\cdot\min\{n, \Delta\})))$ time and $\mathcal{O}(nm(\Delta(d+1)+1))$ space.*

Proof. Correctness: For each integer $1 \leq i \leq n, 0 \leq j \leq \Delta, 0 \leq k \leq \min\{i, m\}$, and $0 \leq p \leq \min\{j, d\}$, all the substitutions and if-conditions in Algorithm 1 follow from Lemmas 2.1, 2.2, 2.3 and 2.4. Furthermore, the values $h[i, j, k_\leq, p_\leq]$, $h[i, j, k_=, p_\leq]$, and $h[i, j, k_=, p_=]$ are computed by the recursive relations given in Lemmas 2.1 and 2.4. This implies that Algorithm 1 correctly computes the required value $h[n, \Delta, m_\leq, d_\leq]$.

Complexity analysis: There are three nested loops over the variables i, j, and p at line 1, which take $\mathcal{O}(n(\Delta(d + 1) + 1))$ time. Following there are five nested loops: over variables i, j, k, p, and q at lines 2, 3, 4, 5, and 15, respectively. The loop at line 2 is of size $\mathcal{O}(n)$, while the loop at line 3 is of size $\mathcal{O}(\Delta)$. Similarly, the loops at lines 4 and 5 are of size $\mathcal{O}(m)$ and $\mathcal{O}(d)$, respectively. The fifth nested loop at line 15 is of size $\mathcal{O}(n)$ (resp., $\mathcal{O}(\min\{n, \Delta\})$) if $p = 0$ (resp., otherwise). Thus from line 2 - 1, Algorithm 1 takes $\mathcal{O}(n^2m)$ (resp., $\mathcal{O}(nm\Delta(n + d \cdot \min\{n, \Delta\})))$ time if $\Delta = 0$ (resp., otherwise). Therefore, Algorithm 1 takes $\mathcal{O}(nm(n + \Delta(n + d \cdot \min\{n, \Delta\})))$ time.

The algorithm stores three four-dimensional arrays. When $\Delta = 0$, for each integer $1 \leq i \leq n$, and $1 \leq k \leq \min\{i, m\}$ we store $h[i, 0, k_\leq, 0_\leq], h[i, 0, k_=, 0_\leq]$ and $h[i, 0, k_=, 0_=]$, taking $\mathcal{O}(nm)$ space. When $\Delta \geq 1$, then for each integer $1 \leq i \leq n$, $0 \leq j \leq \Delta$, $1 \leq k \leq \min\{i, m\}$ and $0 \leq p \leq \min\{j, d\}$ we store $h[i, j, k_\leq, p_\leq], h[i, j, k_=, p_\leq]$ and $h[i, j, k_=, p_=]$, taking $\mathcal{O}(nm\Delta(d + 1))$ space. Hence, Algorithm 1 takes $\mathcal{O}(nm(\Delta(d + 1) + 1))$ space. □

Theorem 2.1. *For any two integers $n \geq 1$ and $\Delta \geq 0$, the number of non-isomorphic graphs with n vertices, Δ self-loops and tree skeleton can be obtained in $\mathcal{O}(n^2(n + \Delta(n + \Delta \cdot \min\{n, \Delta\})))$ time and $\mathcal{O}(n^2(\Delta^2 + 1))$ space.*

Proof. By Jordan [41], we can uniquely consider any tree as a rooted tree by either regarding its unicentroid as the root, or in the case of a bicentroid, by introducing a virtual vertex on the bicentroid and assuming the virtual vertex as the root of the tree. By the definition of a unicentroid, the number of mutually non-isomorphic graphs with n vertices, Δ self-loops, tree skeleton and a unicentroid is $h(n, \Delta, \lfloor (n-1)/2 \rfloor_{\leq}, \Delta_{\leq})$. Further, if n is even, then there exist trees with n vertices and a bicentroid. This implies that the number of mutually non-isomorphic graphs with n vertices, Δ self-loops and tree skeleton is $h(n, \Delta, \lfloor (n-1)/2 \rfloor_{\leq}, \Delta_{\leq})$ when n is odd. Let n be an even integer. Then any graph H with n vertices, Δ self-loops, tree skeleton and a bicentroid has two connected components, A and B obtained by the removal of the bicentroid such that $A \in \mathcal{H}(n/2, i, n/2 - 1_{\leq}, i_{\leq})$ and $B \in \mathcal{H}(n/2, \Delta - i, n/2 - 1_{\leq}, \Delta - i_{\leq})$ for some $0 \leq i \leq \lfloor \Delta/2 \rfloor$, where if Δ is even then for $i = \Delta/2$, both of the components A and B belong to $\mathcal{H}(n/2, \Delta/2, n/2 - 1_{\leq}, \Delta/2_{\leq})$.

Note that for any $0 \leq i \leq \lfloor (\Delta - 1)/2 \rfloor$, it holds that

$$\mathcal{H}(n/2, i, n/2 - 1_{\leq}, i_{\leq}) \cap \mathcal{H}(n/2, \Delta - i, n/2 - 1_{\leq}, \Delta - i_{\leq}) = \emptyset.$$

Therefore, when Δ is odd (resp., even), the number of mutually non-isomorphic graphs with n vertices, Δ self-loops, tree skeleton and a bicentroid is

$$\sum_{i=0}^{\lfloor (\Delta-1)/2 \rfloor} h(n/2, i, n/2 - 1_{\leq}, i_{\leq})\ h(n/2, \Delta - i, n/2 - 1_{\leq}, \Delta - i_{\leq}) + \alpha \binom{h(n/2, \Delta/2, n/2 - 1_{\leq}, \Delta/2_{\leq}) + 1}{2},$$

such that $\alpha = 0$ (resp., $\alpha = 1$). Thus, the number of mutually non-isomorphic graphs with n vertices, Δ self-loops and tree skeleton is

$$h(n, \Delta, \lfloor (n-1)/2 \rfloor_{\leq}, \Delta_{\leq}) + \sum_{i=0}^{\lfloor (\Delta-1)/2 \rfloor} h(n/2, i, n/2 - 1_{\leq}, i_{\leq})\ h(n/2, \Delta - i, n/2 - 1_{\leq}, \Delta - i_{\leq}) + \alpha \binom{h(n/2, \Delta/2, n/2 - 1_{\leq}, \Delta/2_{\leq}) + 1}{2} \tag{2.3.6}$$

such that $\alpha = 0$ (resp., $\alpha = 1$) when Δ is odd (resp., even). Moreover, for each $0 \leq i \leq \Delta$, Algorithm 1 also computes and stores $h(n/2, i, n/2 - 1_{\leq}, i_{\leq})$ during the calculation of $h(n, \Delta, \lfloor (n-1)/2 \rfloor_{\leq}, \Delta_{\leq})$, and therefore the required result follows from Lemma 2.6. □

2.4 Results and Application

We implemented the proposed DP algorithm and counted graphs with a given number of vertices, self-loops and tree skeleton. The experiments were performed on a PC with an Intel Core i7-500 processor, running at 2.70 GHz, 16 GB of memory, and Windows 10. The results are listed in Table 2.1 from which it is evident that the proposed method efficiently counts graphs with n vertices, Δ self-loops and tree skeleton.

Table 2.1. Experimental results of the counting algorithm.

(n, Δ)	Number of Graphs	Time (sec.)
$(10, 0)$	106	0.000173
$(20, 0)$	823,065	0.00048
$(10, 5)$	91,037	0.001193
$(10, 30)$	6,629,790,712	0.00881
$(20, 10)$	5,143,681,226,004	0.006869
$(30, 10)$	2,547,562,522,909,694,331	0.015901

We next give a lower bound and an upper bound on the number of tree-like polymer topologies with self-loops of a given cycle rank. For this we prove the following results.

Lemma 2.7. *For an integer $n \geq 2$, there exists at least one tree-like polymer with n vertices and Δ self-loops if $\Delta \geq \lceil \frac{n}{2} \rceil + 1$.*

Proof. Consider a tree T of n vertices of diameter $\lfloor \frac{n}{2} \rfloor$ such that T contains a path of length $\lfloor \frac{n}{2} \rfloor$, in which each non-end vertex has degree at least 3. Observe that when n is even, the tree T has exactly $\lceil \frac{n}{2} \rceil - 1$ vertices of degree 3, and hence $n - \lceil \frac{n}{2} \rceil + 1 = \lceil \frac{n}{2} \rceil + 1$ vertices of degree less than 3. When n is odd, the tree T has $\lceil \frac{n}{2} \rceil - 3$ vertices of degree 3 and one vertex of degree 4. Thus, in this case, the number of vertices of degree less than 3 is $n - \lceil \frac{n}{2} \rceil + 2 = \left(2 \lceil \frac{n}{2} \rceil - 1\right) - \lceil \frac{n}{2} \rceil + 2 = \lceil \frac{n}{2} \rceil + 1$. This implies that T can be transformed into a polymer with $\lceil \frac{n}{2} \rceil + 1$ self-loops by assigning a self-loop to each vertex of degree less than 3. Hence, $\lceil \frac{n}{2} \rceil + 1$ self-loops are sufficient to get a tree-like polymer with n vertices. $\square$

For two integers $n \geq 1$ and $\Delta \geq 0$, let $g(n, \Delta)$ denote the number of graphs with n vertices, Δ self-loops and tree skeleton. For $r \geq 1$, let $p(r)$ denote the number of tree-like polymers of rank r with self-loops and no multi-edges. Observe that a graph with n vertices, k self-loops and tree skeleton at each vertex is a polymer with n vertices of cycle rank kn. From this fact and Lemma 2.7 it holds that

$$\sum_{n,k\in\mathbb{Z}^+:nk=r} g(n,0) \le p(r) \le \sum_{n\in\mathbb{Z}^+:\lceil \frac{n}{2}\rceil+1\le r} g(n,r). \tag{2.4.7}$$

2.5 Concluding Remarks

We presented an efficient method to count the number of all mutually non-isomorphic graphs with a given number of vertices, self-loops and tree skeleton. The proposed method is based on dynamic programming where we count the number of all mutually non-isomorphic rooted graphs with a given number n of vertices, Δ self-loops and tree skeleton in $\mathcal{O}(n^2(n+\Delta(n+\Delta\cdot\min\{n,\Delta\})))$ time and $\mathcal{O}(n^2(\Delta^2+1))$ space. As an application of our results, we gave lower and upper bounds on the number of tree-like polymer topologies with a given cycle rank. This is an interesting application of DP to objects such as trees, and offers the advantage of getting the size of the entire solution space at low computational complexity without explicitly generating each object.

An interesting direction for future research is to efficiently count all mutually non-isomorphic tree-like polymer topologies with a given number of vertices and self-loops.

3 Enumerating Tree-Like Graphs with a Given Cycle Rank

3.1 Introduction

For an enumeration method it is necessary to generate all possible required structures without duplication in low computational complexity, due to which designing an enumeration method is not an easy task. Several methods has been developed to generate chemical graphs. These methods are mainly based on the branching algorithm paradigm; the required chemical compounds appear at the leaves of a computation tree. However, these algorithms generate many invalid intermediate structures that appear at the non-leaf nodes of the computation tree [40]. Due to this fact, these methods are inefficient to generate chemical compounds with more than 20 non-hydrogen atoms. Thus, it is natural to explore and develop such methods that can enumerate chemical compounds without generating invalid intermediate structures. Jin et al. [40] proposed one such chemical compound generation method based on the junction tree and the variational autoencoder.

Recall that the polymer topology of a chemical compound is a connected multi-graph where all vertices have degree at least three. Polymer topologies P are often classified with respect to their cycle rank, which is the number of edges that are necessary to remove to get a spanning tree of P. The class of graphs with a tree skeleton, $\Delta \geq 0$ self-loops, and no multiple edges contains all tree-like polymer topologies with cycle rank Δ.

Recently, Azam et al. [8] proposed a method to count all trees with given numbers of vertices and self-loops by using dynamic programming. As a result, they gave the upper bound and the lower bound on the number of tree-like mutually non-isomorphic polymer topologies with a given rank.

In this chapter we develop an efficient method to enumerate all mutually non-isomorphic graphs with a tree skeleton, n vertices, Δ self-loops, and no multiple edges without generating invalid intermediate structures. The idea of our method is to define a canonical representation of rooted graphs with the said structures and then enumerate these graphs by generating their canonical representations. As a consequence of our method, we can get all polymer topologies with a tree skeleton, a given cycle rank, self-loops, and no multiple edges.

We organize this chapter as follows: In Section 3.2, we discuss some preliminaries. In Section 3.3, we first prove the mathematical properties based on which we develop our enumeration method. We discuss experimental results and an application of our enumeration method to generate all polymer topologies with a tree skeleton and a given cycle rank in Section 3.4. We conclude and discuss some future directions in Section 3.5.

3.2 Preliminaries

Let $n \geq 1$ and $\Delta \geq 0$ be two integers and H be a rooted graph in $\mathcal{H}(n, \Delta)$. An *ordered graph* (H, π) of H is defined to be the rooted graph H with a left-to-right ordering π on the children of each vertex of the rooted skeleton $\gamma(H)$. Let $K = (H, \pi)$ be an ordered graph of H. For a vertex $v \in V(K)$, we define the ordered subgraph K_v of K to be a subgraph of K rooted at v induced by v and its descendants in the rooted skeleton $\gamma(K)$ with preserving the ordering π on the children of each vertex in $\gamma(K_v)$. For a vertex $v \in N_K(r_K)$, we call the ordered subgraph K_v an ordered root-subgraph of K.

For an ordered tree, we discuss two vertex orderings: depth first search (DFS) ordering [25] and sibling-depth first search (SDFS) ordering. In DFS ordering, we index the vertices of a given ordered tree starting from the root and visiting them from left to right. Masui et al. [46] introduced the SDFS ordering for simple ordered trees. For an ordered tree $T = (L, \pi)$ with n vertices and a left-to-right ordering π, the SDFS ordering is defined to be a vertex ordering obtained by indexing the vertices from the set $\{1, 2, \ldots, n\}$ such that:

(i) The root has index one;

(ii) All siblings are indexed consecutively according to the left-to-right ordering π; and

(iii) All descendants of a vertex v are indexed consecutively with indices larger than that of v and smaller than the indices of the descendants of any vertex u, which is not a descendant of v with index larger than v.

Examples of an ordered tree and its vertex indexing in DFS and SDFS ordering are illustrated in Figure 3.1.

Let $A = (a_1, a_2, \ldots, a_n)$ and $B = (b_1, b_2, \ldots, b_m)$ be two sequences over integers. We say that the sequence A is lexicographically smaller $A \prec B$ than the

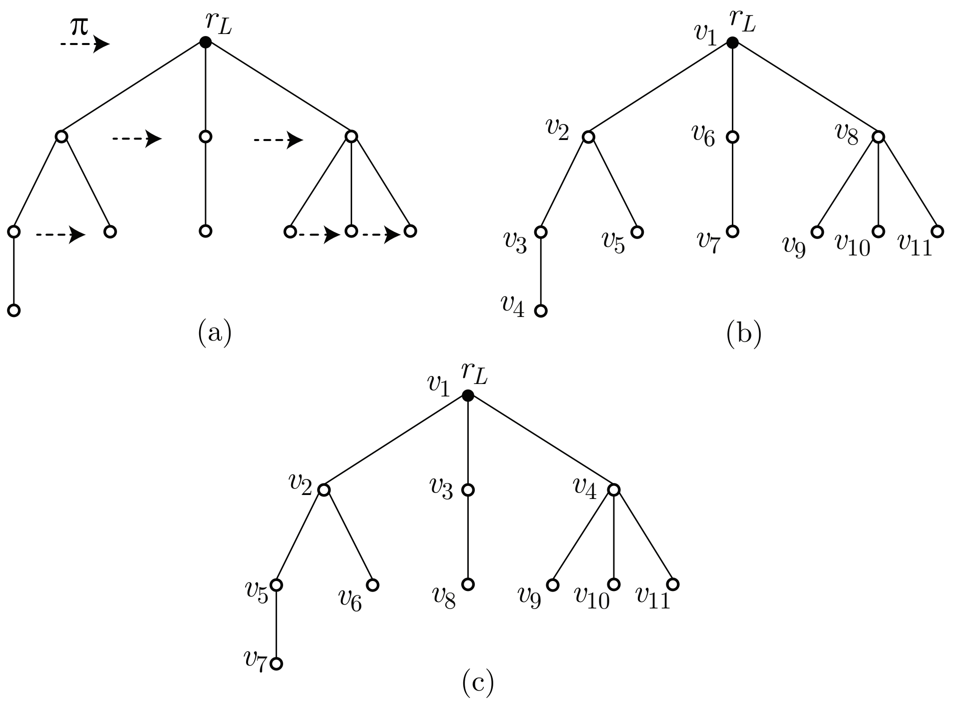

π
r_L
(a)
v_1 r_L
v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}
(b)
v_1 r_L
v_2 v_3 v_4 v_5 v_6 v_7 v_8 v_9 v_{10} v_{11}
(c)

Figure 3.1. Examples of an ordered tree and vertex indexing: (a) an ordered tree $T = (L, \pi)$ with left-to-right ordering π indicated by the dashed arrow; (b) ordered tree $T = (L, \pi)$ from (a) with vertices indexed in depth first search (DFS) order; and (c) ordered tree $T = (L, \pi)$ from (a) with vertices indexed in sibling-depth first search (SDFS) order.

sequence B if there exists an integer ℓ, $1 \leq \ell \leq \min\{n, m\}$, such that for each integer i, $1 \leq i \leq \ell$, it holds that $a_i = b_i$ and

(i) either $\ell = n$ with $n < m$ or

(ii) $\ell < \min\{n, m\}$ with $a_{\ell+1} < b_{\ell+1}$.

In such a case, we say that the sequence B is lexicographically greater $B \succ A$ than the sequence A. We define the concatenation $A \oplus B$ of the sequences A and B to be the sequence $(a_1, \ldots, a_n, b_1, \ldots, b_m)$.

3.3 Enumerating Tree-Like Graphs with a Given Number of Vertices and Self-loops

For two integers $n \geq 1$ and $\Delta \geq 0$, the aim of this section is to present a method to generate all rooted graphs in $\mathcal{H}(n, \Delta)$. The idea of our enumeration method is to generate a rooted graph $H \in \mathcal{H}(n, \Delta)$ by generating a canonical ordered graph of H. To achieve this, we define a canonical graph of a rooted graph $H \in \mathcal{H}(n, \Delta)$ and represent the canonical ordered graph with a sequence by using its ordered subgraphs. Finally, generate the canonical ordered graph of a rooted graph by using the sequence representation of the canonical ordered graph.

We next present a canonical representation of rooted graphs in $\mathcal{H}(n, \Delta)$ based on a generalization of the canonical representation of simple rooted trees with n vertices introduced by Masui et al. [46]. Recall that $\mathcal{H}(n, 0)$ denote a maximal set of all mutually rooted non-isomorphic simple rooted trees with n vertices. Further, note that for $\Delta \geq 1$, it is necessary for a canonical representation of a rooted graph $H \in \mathcal{H}(n, \Delta)$ to contain the information of vertices and self-loops in H.

Let H be a rooted graph in $\mathcal{H}(n, \Delta)$ and r_H denote its root. Further, let $K = (H, \pi)$ be an ordered graph of H with a left-to-right ordering π. For an integer $i \in [1, n]$ and i-th vertex v_i of K following the SDFS ordering on the rooted skeleton $\gamma(K)$, let $K_{(i)}$ denote the ordered subgraph K_{v_i} of K for convenience.

We introduce a canonical representation of K by using the information of the number of vertices and self-loops in the ordered subgraphs of K. For the vertices $\{v_1, v_2, \ldots, v_n\}$ of K indexed by SDFS ordering on the rooted skeleton $\gamma(K)$, we define the sequence representation $\mathrm{SR}(K)$ of K to be a sequence of the size of each ordered subgraph $K_{(i)}$, integer $i \in [2, n]$, of K:

$$\mathrm{SR}(K) \triangleq (\mathrm{s}(K_{(2)}), \mathrm{s}(K_{(3)}), \ldots, \mathrm{s}(K_{(n)})). \tag{3.3.1}$$

Examples of a rooted graph $H \in \mathcal{H}(11, 3)$, an ordered graph $K = (H, \pi)$ of H with a left-to-right ordering π, and vertices indexed in SDFS ordering and canonical representation $\mathrm{SR}(K)$ of K are illustrated in Figure 3.2(a)–(c).

The next lemma states that the sequence representation of an ordered graph K is a concatenation of

(i) A sequence of the size of the root-subgraphs of K in the left-to-right ordering and

(ii) The sequence representation of all root-subgraphs of K following the left-to-right ordering.

Lemma 3.8. *Let K be an ordered graph with $n \geq 1$ vertices and $\Delta \geq 0$ self-loops. For integers $d = \deg_K(r_K)$ and $i \in [1, d]$, let K_i denote the i-th root-subgraph of K in the left-to-right ordering. Then, it holds that:*

$$\mathrm{SR}(K) = (\mathrm{s}(K_1), \ldots, \mathrm{s}(K_d)) \oplus \mathrm{SR}(K_1) \oplus \cdots \oplus \mathrm{SR}(K_d).$$

Proof. We know that in SDFS ordering, the root vertex is indexed by one, and the siblings of a vertex are indexed consecutively. This implies that the subsequence of the first d entries of $\mathrm{SR}(K)$ is equal to $(\mathrm{s}(K_1), \ldots, \mathrm{s}(K_d))$. Furthermore, for an integer $i = 2$ (resp., $i \in [3, d+1]$), the SDFS ordering assigns the index to the descendants of the i-th vertex consecutively and greater than the indices of the children (resp., descendants) of the $(i-1)$-th vertex. From this, it follows that for an integer $i = 2$ (resp., $i \in [3, d+1]$), the entries of $\mathrm{SR}(K_{(i)})$ appear consecutively after the entries of the sequence $\mathrm{s}(K_1), \ldots, \mathrm{s}(K_d)$ (resp., $\mathrm{SR}(K_{(i-1)})$) in $\mathrm{SR}(K)$. This implies that for an integer $i \in [1, d]$, the subsequence of $\mathrm{SR}(K)$ consisting of $\mathrm{n}(K_i) - 1$ consecutive entries starting from $d + \sum_{1 \le h \le i-1} \mathrm{n}(K_h) - (i-1) + 1$ is actually $\mathrm{SR}(K_i)$. Hence, it follows that $\mathrm{SR}(K) = (\mathrm{s}(K_1), \ldots, \mathrm{s}(K_d)) \oplus \mathrm{SR}(K_1) \oplus \cdots \oplus \mathrm{SR}(K_d)$. □

For the ordered graph K in Figure 3.2(c), $\mathrm{SR}(K) = ((4,1), (2,0), (4,2), (1,0), (1,1), (1,0), (1,0), (2,0), (1,1), (1,0))$ and $\deg_K(r_K) = 3$. This implies that K has three root-subgraphs K_1, K_2 and K_3 following the left-to-right ordering. From Figure 3.2(c), we have $(\mathrm{s}(K_1), \mathrm{s}(K_2), \mathrm{s}(K_3)) = ((4,1), (2,0), (4,2))$, $\mathrm{SR}(K_1) = ((1,0), (1,1), (1,0))$, $\mathrm{SR}(K_2) = ((1,0))$, and $\mathrm{SR}(K_3) = ((2,0), (1,1), (1,0))$. Thus, we see that $\mathrm{SR}(K) = (\mathrm{s}(K_1), \mathrm{s}(K_2), \mathrm{s}(K_3)) \oplus \mathrm{SR}(K_1) \oplus \mathrm{SR}(K_2) \oplus \mathrm{SR}(K_3)$.

We rephrase the recursion in Lemma 3.8 for a sequence of pairs in the following paragraph and claim that this recursion is a sufficient condition for a sequence of pairs to be the sequence representation of some ordered graph. We prove this claim in Theorem 3.2.

Let $n \ge 1$ and $\Delta \ge 0$ be two integers and $M = ((a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1}))$ be a sequence of pairs of integers with $a_i \ge 1$ and $b_i \ge 0$, integer $i \in [1, n-2]$. We say that the sequence M is (n, Δ)-admissible if either $n = 1$ or

(i) there exists an integer $d \in [1, n-1]$ such that $n - 1 = \sum_{1 \le i \le d} a_i$ with $\Delta \ge \sum_{1 \le i \le d} b_i$ and

(ii) for each integer $i \in [1, d]$, the subsequence of M consisting of $a_i - 1$ consecutive entries starting from $d + \sum_{1 \le h \le i-1} a_h - (i-1) + 1$ is (a_i, b_i)-admissible.

Theorem 3.2. *Let $n \ge 1$ and $\Delta \ge 0$ be two integers and $M = ((a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1}))$ be a sequence of pairs of integers with $a_i \ge 1$ and $b_i \ge 0$, integer $i \in [1, n-2]$.*

(i) *Sequence M is the sequence representation $\mathrm{SR}(K)$ of some ordered graph K with n vertices and Δ self-loops if and only if M is (n, Δ)-admissible.*

(ii) *Whether M is admissible or not can be tested in $\mathcal{O}(n)$ time.*

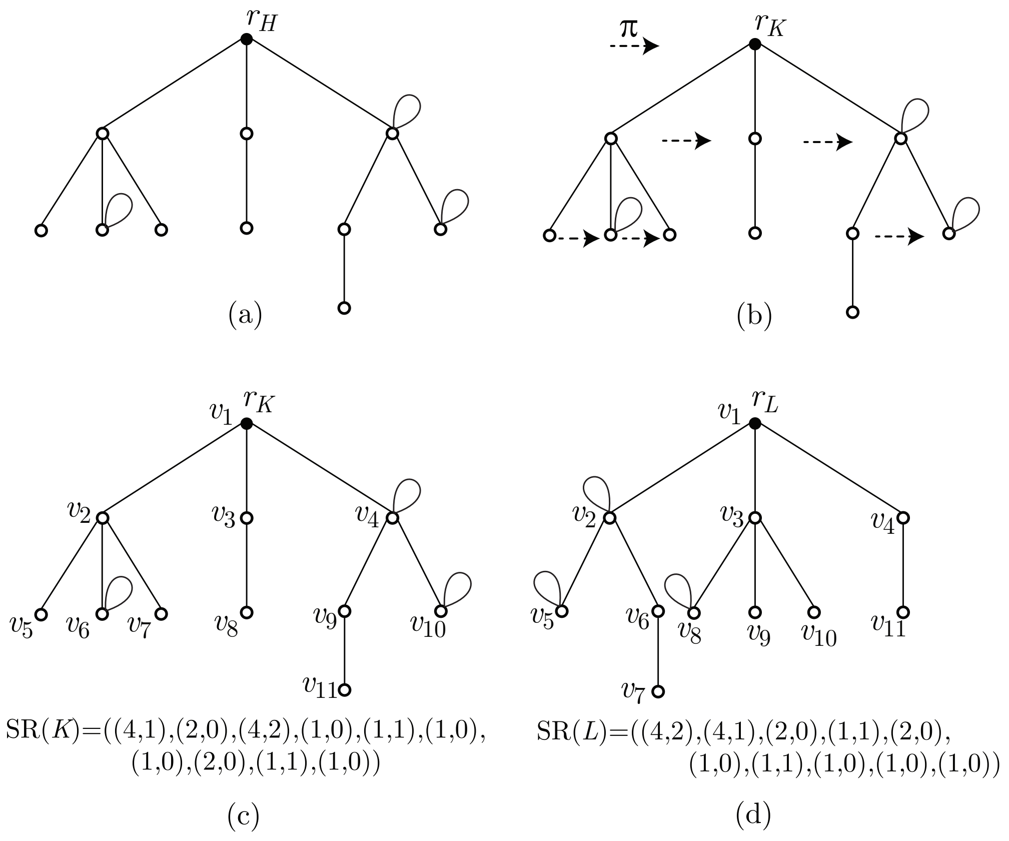

r_H
(a)
π
r_K
(b)
v_1 r_K
v_2 v_3 v_4
v_5 v_6 v_7 v_8 v_9 v_{10}
v_{11}
SR(K)=((4,1),(2,0),(4,2),(1,0),(1,1),(1,0),(1,0),(2,0),(1,1),(1,0))
(c)
v_1 r_L
v_2 v_3 v_4
v_5 v_6 v_8 v_9 v_{10} v_{11}
v_7
SR(L)=((4,2),(4,1),(2,0),(1,1),(2,0),(1,0),(1,1),(1,0),(1,0),(1,0))
(d)

Figure 3.2. (a) A rooted graph $H \in \mathcal{H}(11,3)$; (b) an ordered graph K of H with left-to-right ordering π on siblings; (c) the ordered graph K of H with SDFS vertex indexing and $\mathrm{SR}(K)$; and (d) the canonical representation of H and the ordered graph with sequence representation equal to the canonical representation of H.

(iii) *When M is (n,Δ)-admissible, the ordered graph K with* $\mathrm{SR}(K) = M$ *can be constructed in $\mathcal{O}(n)$ time.*

Proof. For sequence M with an integer $d \in [1, n-1]$ such that $n - 1 = \sum_{1\le i\le d} a_i$ and $\Delta \ge \sum_{1\le i\le d} b_i$, let M_i denote the subsequence of M consisting of $a_i - 1$ consecutive entries starting from $d + \sum_{1\le h\le i-1} a_h - (i-1) + 1$. For an ordered graph K and integer $i \in [1, \deg_K(r_K)]$, let K_i denote the i-th root-subgraph of K in the left-to-right ordering.

(i) The if-part: Suppose that $M = \mathrm{SR}(K)$ for some ordered graph K with n vertices and Δ self-loops. If $n = 1$, then M is $(1,\Delta)$-admissible by the definition of admissibility. Let us assume that $n \ge 2$. Then, for $d = \deg_K(r_K)$, it holds that $n - 1 = \sum_{1\le i\le d} \mathrm{n}(K_i)$ and $\Delta \ge \sum_{1\le i\le d} s(K_i)$. Further, by Lemma 3.8 for each integer $i \in [1, d]$, the subsequence M_i of M is equal to $\mathrm{SR}(K_i)$. Thus, by recursively using Lemma 3.8 for $\mathrm{SR}(K_i)$, $i \in [1, d]$, we see that the sequence representation $\mathrm{SR}(K_i)$ is $(\mathrm{n}(K_i), s(K_i))$-admissible. Hence, it follows that the sequence representation $\mathrm{SR}(K)$ is (n, Δ)-admissible.

The only-if part: We prove the converse of (i) by induction on n.

For $n = 1$, M is $(1, \Delta)$-admissible by the definition of admissibility. Note that M is an empty sequence in this case. Let K be an ordered graph with $n = 1$ vertices and Δ self-loops. Then, $\mathrm{SR}(K)$ is an empty sequence, and hence, $M = \mathrm{SR}(K)$.

Suppose that the converse of (i) holds for any positive integer ℓ. We show that the converse holds for the integer $\ell + 1$. Let M be $(\ell + 1, \Delta)$-admissible. Then, by the definition of admissibility, there exists an integer $d \in [1, \ell]$ such that $\ell = \sum_{1\le i\le d} a_i$ and $\Delta \ge \sum_{1\le i\le d} b_i$. This implies that for an integer $i \in [1, d]$, we have $a_i \le \ell$. Further, for each integer $i \in [1, d]$, the subsequence M_i is (a_i, b_i)-admissible by the admissibility of M. This and the inductive hypothesis that the converse of (i) holds for any integer $\ell \ge 1$ imply that for each integer $i \in [1, d]$, there exists an ordered graph H with a_i vertices and b_i self-loops such that $M_i = \mathrm{SR}(H)$. Let K denote the ordered graph with $\ell + 1$ vertices, Δ self-loops, $\deg_K(r_K) = d$, $\Delta - \sum_{1\le i\le d} b_i$ self-loops on the root r_K, and the i-th root subgraph K_i of K be the ordered subgraph H such that $M_i = \mathrm{SR}(H)$. Then, it immediately follows that

$$\mathrm{SR}(K) = (\mathrm{s}(K_1), \dots, \mathrm{s}(K_d)) \oplus M_1 \oplus M_2 \oplus \dots \oplus M_d.$$

This means that $M = \mathrm{SR}(K)$ holds, since $((a_1, b_1), \dots, (a_d, b_d)) = (\mathrm{s}(K_1), \dots, \mathrm{s}(K_d))$, showing that the converse holds for the integer $\ell + 1$.

Hence, by mathematical induction, the converse of (i) holds for any integer $n \ge 1$.

(ii) We prove this result by induction on n.

For $n = 1$, the sequence M is an empty sequence and is $(1, \Delta)$-admissible by the definition of admissibility. Therefore, it takes constant $\mathcal{O}(1)$ time to test admissibility in this case.

Suppose that for $n = \ell$, $\ell \ge 1$, the admissibility of sequence M can be tested in $\mathcal{O}(\ell)$ time. We show that the statement (ii) holds for $n = \ell + 1$. To show if M is $(\ell + 1, \Delta)$-admissible, we need to find an integer $d \in [1, \ell]$ such that $\ell = \sum_{1\le i\le d} a_i$ and $\Delta \ge \sum_{1\le i\le d} b_i$. Such an integer d can be identified in $\mathcal{O}(d)$ time. Suppose that such an integer d exists for M. Then, for each integer $i \in [1, d]$, we next need to test if the subsequence M_i is (a_i, b_i)-admissible. Note that the size of the sequence M_i is $a_i - 1$, $i \in [1, d]$. By $\ell = \sum_{1\le i\le d} a_i$, it holds that $a_i \le \ell$, $i \in [1, d]$. This and the inductive hypothesis imply that for an integer $i \in [1, d]$, the admissibility of the sequence M_i can be tested in $\mathcal{O}(a_i)$ time. Thus, the time testing admissibility of M is $\mathcal{O}(d+\sum_{1\leq i\leq d} a_i) = \mathcal{O}(d+\ell) = \mathcal{O}(\ell+1)$, since $d \leq \ell$.

Hence, by mathematical induction, the admissibility of a sequence M of size $n-1$, $n \geq 1$ can be tested in $\mathcal{O}(n)$ time.

(iii) We prove the claim in (iii) by induction on n.

For $n = 1$, M is $(1, \Delta)$-admissible and is the sequence representation of the ordered graph K with only one vertex and Δ self-loops. This implies that K can be constructed in $\mathcal{O}(1)$ time.

Suppose that for $n = \ell$, $\ell \geq 1$, the statement (iii) holds. We show that the statement (iii) holds for $n = \ell + 1$. Let M be an $(\ell + 1, \Delta)$-admissible sequence. Then, there exists an integer $d \in [1, \ell]$, such that $\ell = \sum_{1\leq i\leq d} a_i$ and $\Delta \geq \sum_{1\leq i\leq d} b_i$. By (i), there exists an ordered graph K with n vertices and Δ self-loops such that $\mathrm{SR}(K) = M$. Thus, it holds that $\deg_K(r_K) = d$. Further, by Lemma 3.8, for each integer $i \in [1, d]$, the i-th root-subgraph K_i of K has a_i vertices and b_i self-loops. Observe that such an integer d exists uniquely due to the admissibility of M. This implies that $\deg_K(r_K)$ can be obtained in $\mathcal{O}(d)$ time.

By $\mathrm{SR}(K) = M$ and Lemma 3.8, for each integer $i \in [1, d]$, it holds that $\mathrm{SR}(K_i) = M_i$. Recall that the size of $\mathrm{SR}(K_i)$ is $\mathrm{n}(K_i) - 1$, which is equal to $a_i - 1$, $i \in [1, d]$. Further, by $\ell = \sum_{1\leq i\leq d} a_i$, it follows that $a_i \leq \ell$, $i \in [1, d]$. Thus, by the inductive hypothesis for an integer $i \in [1, d]$, the subgraph K_i can be constructed from M_i in $\mathcal{O}(a_i)$ time. Since $d + \sum_{1\leq i\leq d} a_i = d + \ell$ and $d \leq \ell$, K can be constructed in $\mathcal{O}(\ell + 1)$ time from M.

Hence, by mathematical induction, for integers $n \geq 1$ and $\Delta \geq 0$ and an (n, Δ)-admissible sequence M, the ordered graph K with $\mathrm{SR}(K) = M$ can be constructed by M in $\mathcal{O}(n)$ time. □

Let $M = ((a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1}))$ be an (n, Δ)-admissible sequence. Then, there exists an integer $d \in [1, n-1]$ such that $n - 1 = \sum_{1\leq i\leq d} a_i$. Furthermore, by Theorem 3.2(i), there exists an ordered graph K with n vertices and Δ self-loops such that $\deg_K(r_K) = d$ and $\mathrm{SR}(K) = M$. Note that such an integer d and ordered graph K are unique. We call the integer d the root-degree of M and denote it by $\mathrm{d}(M)$. Moreover, for each integer $i \in [1, \mathrm{d}(M)]$, the subsequence of M consisting of $a_i - 1$ consecutive entries starting from $\mathrm{d}(M) + \sum_{1\leq h\leq i-1} a_h - (i-1) + 1$ is equal to the sequence representation of the i-th root-subgraph of M. For an integer $i \in [1, \mathrm{d}(M)]$, we call such a subsequence of M the i-th root-subsequence of M and denote it by $M(i)$.

By Theorem 3.2(i), it follows that an ordered graph K with $n \geq 1$ vertices and $\Delta \geq 0$ self-loops can be completely determined by $\mathrm{SR}(K)$. Thus, we define a canonical representation of a rooted graph $H \in \mathcal{H}(n, \Delta)$ as follows. For a rooted graph $H \in \mathcal{H}(n, \Delta)$, we define the canonical representation to be an (n, Δ)-admissible sequence M such that M is lexicographically maximum among all (n, Δ)-admissible sequences that are the sequence representation of ordered graphs of H.

In Figure 3.2(d), we show the canonical representation M of the rooted graph $H \in \mathcal{H}(11, 3)$ illustrated in Figure 3.2(a). Further, we show the ordered graph L such that $\mathrm{SR}(L) = M$.

To generate all rooted graphs in $\mathcal{H}(n, \Delta)$, it is enough to generate the canonical representation of each rooted graph $H \in \mathcal{H}(n, \Delta)$ by Theorem 3.2(i). For two integers $n \geq 1$ and $\Delta \geq 0$, let $\mathcal{M}(n, \Delta)$ denote the set of all (n, Δ)-admissible sequences that are canonical representation of graphs in $\mathcal{H}(n, \Delta)$. Note that the empty sequence is the only sequence in $\mathcal{M}(1, \Delta)$. In the next lemma, we give a characterization of sequences in $\mathcal{M}(n, \Delta)$.

Lemma 3.9. *Let $n \geq 2$ and $\Delta \geq 0$ be two integers. Let $M = ((a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1}))$ be a sequence of integer pairs with an integer $d \in [1, n-1]$ such that $n-1 = \sum_{1 \leq i \leq d} a_i$. For an integer $i \in [1, d]$, let $M(i)$ denote the subsequence of M consisting of $a_i - 1$ consecutive entries starting from $d + \sum_{1 \leq h \leq i-1} a_h - (i-1) + 1$. Then, $M \in \mathcal{M}(n, \Delta)$ if and only if the following hold:*

(i) *$a_i \geq 1, \forall i \in [1, d]$, $\sum_{1 \leq i \leq d} a_i = n - 1$ and $a_i \geq a_{i+1}, \forall i \in [1, d-1]$;*

(ii) *$b_i \geq 0, \forall i \in [1, d]$, $\sum_{1 \leq i \leq d} b_i \leq \Delta$ and for each integer $i \in [1, d-1]$ such that $a_i = a_{i+1}$, it holds that $b_i \geq b_{i+1}$; and*

(iii) *$M(i) \in \mathcal{M}(a_i, b_i), \forall i \in [1, d]$, and for each integer $i \in [1, d-1]$ such that $a_i = a_{i+1}$ and $b_i = b_{i+1}$, it holds that $M(i) \succeq M(i+1)$.*

Proof. The if part: Let $M \in \mathcal{M}(n, \Delta)$. Then, by the definition of admissibility, it holds that $d = \mathrm{d}(M)$. Let H denote the ordered graph with n vertices and Δ self-loops such that $\mathrm{SR}(H) = M$.

(i) By the admissibility of M, we have $a_i \geq 1, \forall i \in [1, \mathrm{d}(M)]$, $\sum_{1 \leq i \leq \mathrm{d}(M)} a_i = n - 1$. Furthermore, M is the canonical representation of H, and therefore, for the sequence representation $((s_1, s'_1), (s_2, s'_2), \ldots, (s_{n-1}, s'_{n-1}))$ of any ordered graph of H, it holds that $(a_1, \ldots, a_{\mathrm{d}(M)}) \succeq (s_1, \ldots, s_{\mathrm{d}(M)})$. This eventually implies that $a_i \geq a_{i+1}, \forall i \in [1, d-1]$.

(ii) By the admissibility of M, it holds that $b_i \geq 0, \forall i \in [1, \mathrm{d}(M)]$, $\sum_{1 \leq i \leq \mathrm{d}(M)} b_i \leq \Delta$. Moreover, for the sequence representation $((s_1, s'_1), (s_2, s'_2), \ldots, (s_{n-1}, s'_{n-1}))$ of any ordered graph of H such that $s_i = a_i, \forall i \in [1, \mathrm{d}(M)]$, it holds that

$(b_1, \ldots, b_{\mathrm{d}(M)}) \succeq (s'_1, \ldots, s'_{\mathrm{d}(M)})$ since M is the canonical representation of H. This implies that for each integer $i \in [1, \mathrm{d}(M) - 1]$ such that $a_i = a_{i+1}$, it holds that $b_i \geq b_{i+1}$.

(iii) We first prove that for each integer $i \in [1, \mathrm{d}(M)]$, it holds that $M(i) \in \mathcal{M}(a_i, b_i)$.

For an integer $i \in [1, \mathrm{d}(M)]$, let H_i denote the i-th root-subgraph of H following the left-to-right ordering. Then, by Lemma 3.8 for each integer $i \in [1, \mathrm{d}(M)]$, it holds that $\mathrm{SR}(H_i) = M(i)$.

Suppose on the contrary that there exists an integer $i \in [1, \mathrm{d}(M)]$ such $M(i) \notin \mathcal{M}(a_i, b_i)$. This means that there exists an ordered graph L that is rooted isomorphic to H_i, and $\mathrm{SR}(L) \succ M(i)$ holds. Let M' denote the sequence obtained from M by replacing the subsequence M_i with $\mathrm{SR}(L)$. Clearly, M' is (n, Δ)-admissible, and $M' \succ M$ holds by the construction of M'. Let H' denote the ordered graph obtained by replacing H_i with L in H and preserving the ordering of the children of each vertex in L. Then, we see that H' is rooted isomorphic to H, and $\mathrm{SR}(H') = M'$. This contradicts the fact that M is a sequence in $\mathcal{M}(n, \Delta)$. Hence, for each integer $i \in [1, \mathrm{d}(M)]$, it holds that $M(i) \in \mathcal{M}(a_i, b_i)$.

Recall that M is the canonical representation of H. This implies that for the sequence representation $((s_1, s'_1), (s_2, s'_2), \ldots, (s_{n-1}, s'_{n-1}))$ of any ordered graph of H such that $(s_i, s'_i) = (a_i, b_i), \forall i \in [1, \mathrm{d}(M)]$, it holds that $((a_{\mathrm{d}(M)+1}, b_{\mathrm{d}(M)+1}), \ldots, (a_{n-1}, b_{n-1})) \succeq ((s_{\mathrm{d}(M)+1}, s'_{\mathrm{d}(M)+1}), \ldots, (s_{n-1}, s'_{n-1}))$. This implies that for each integer $i \in [1, \mathrm{d}(M)-1]$ such that $a_i = a_{i+1}$ and $b_i = b_{i+1}$, we have $M(i) \succeq M(i+1)$.

The only-if part: Let M satisfy (i), (ii), and (iii). We show that $M \in \mathcal{M}(n, \Delta)$. To prove this, we show that M is a canonical representation of some graph in $\mathcal{H}(n, \Delta)$.

By (i) and (ii), we have $a_i \geq 1, b_i \geq 0, \forall i \in [1, d]$, $n - 1 = \sum_{1 \leq i \leq d} a_i$ and $\Delta \geq \sum_{1 \leq i \leq d} b_i$. Furthermore, for each integer $i \in [1, d]$, the sequence M_i is (a_i, b_i)-admissible, since $M(i) \in \mathcal{M}(a_i, b_i)$ by (iii). This implies that M is (n, Δ)-admissible.

By Theorem 3.2(i), there exists a unique ordered graph $K = (H, \pi)$ such that $\mathrm{SR}(K) = M$ for some $H \in \mathcal{H}(n, \Delta)$. This implies that $\deg_K(r_K) = d$, and for each integer $i \in [1, d]$, the i-th root subgraph of K has a_i vertices and b_i self-loops. This implies that any ordered graph L that is rooted isomorphic to K has x vertices and y self-loops such that $(x, y) = (a_i, b_i)$ for some $i \in [1, d]$.

The condition $a_i \geq a_{i+1}, \forall i \in [1, d-1]$ in (i) implies that for the sequence representation $S = ((s_1, s'_1), (s_2, s'_2), \ldots, (s_{n-1}, s'_{n-1}))$ of any ordered graph that is rooted isomorphic to K, it holds that $M \succeq S$ since $(a_1, \dots, a_d) \succeq (s_1, \dots, s_d)$. For the condition for each integer $i \in [1, d-1]$ such that $a_i = a_{i+1}$, it holds that $b_i \geq b_{i+1}$ in (ii) implies that for the sequence representation $S = ((s_1, s'_1), (s_2, s'_2), \dots, (s_{n-1}, s'_{n-1}))$ of any ordered graph that is rooted isomorphic to K such that $s_i = a_i, \forall i \in [1, d]$, it holds that $M \succeq S$ since $(b_1, \dots, b_d) \succeq (s'_1, \dots, s'_d)$. Finally, for the condition for each integer $i \in [1, d-1]$ such that $a_i = a_{i+1}$ and $b_i = b_{i+1}$, it holds that $M(i) \succeq M(i+1)$ in (iii) implies that for the sequence representation $S = ((s_1, s'_1), (s_2, s'_2), \dots, (s_{n-1}, s'_{n-1}))$ of any ordered graph that is rooted isomorphic to K and $(s_i, s'_i) = (a_i, b_i), \forall i \in [1, d]$, it holds that $M \succeq S$ since $((a_{d+1}, b_{d+1}), \dots, (a_{n-1}, b_{n-1})) \succeq ((s_{d+1}, s'_{d+1}), \dots, (s_{n-1}, s'_{n-1}))$. This eventually implies that M is the canonical representation of H from which it follows that $M \in \mathcal{M}(n, \Delta)$. □

We next give the structure of the sequences that are lexicographically minimum and maximum among all sequences in $\mathcal{M}(n, \Delta)$.

Lemma 3.10. *Let $n \geq 2$ and $\Delta \geq 0$ be two integers.*

(i) *The sequences $N = ((1, 0), (1, 0), \dots, (1, 0))$ and $M = ((n-1, \Delta), (n-2, \Delta), \dots, (1, \Delta))$ each of length $n-1$ are lexicographically minimum and maximum among all sequences in $\mathcal{M}(n, \Delta)$, respectively.*

(ii) *Whether a sequence in $\mathcal{M}(n, \Delta)$ is lexicographically minimum or maximum among all sequences in $\mathcal{M}(n, \Delta)$ can be tested in $\mathcal{O}(n)$.*

Proof. (i) It is easy to observe that the sequence N is (n, Δ)-admissible. Furthermore, for two integers $i \geq 2$ and $j \geq 0$, the ranges of the first and the second entries in any sequence in $\mathcal{M}(i, j)$ are $[1, i-1]$ and $[0, j]$, respectively. This implies that the sequence N is lexicographically minimum among all the sequences in $\mathcal{M}(n, \Delta)$. Moreover, the sequence $((1, \Delta))$ is admissible and lexicographically maximum among all the sequences in $\mathcal{M}(2, \Delta)$. From this, it follows that the sequence $((2, \Delta), (1, \Delta))$ is admissible and lexicographically maximum among all the sequences in $\mathcal{M}(3, \Delta)$. Thus, by using this inductive argument, we can conclude that the sequence M is admissible and lexicographically maximum among all the sequences in $\mathcal{M}(n, \Delta)$.

(ii) We know that a sequence S in $\mathcal{M}(n, \Delta)$ is of length $n-1$, and therefore, by using a for-loop of size $n-1$ and (i), we can test if S is lexicographically minimum or maximum among all sequences in $\mathcal{M}(n, \Delta)$ in $\mathcal{O}(n)$ time. □

Let $\mathrm{S}(n, \Delta)$ and $\mathrm{L}(n, \Delta)$ denote the lexicographically minimum and maximum sequences among all sequences in $\mathcal{M}(n, \Delta)$, respectively.

For a sequence $M \in \mathcal{M}(n, \Delta)$ such that $M \neq \mathrm{S}(n, \Delta)$, we define the predecessor $\mathrm{P}(M)$ of M to be the sequence that is lexicographically maximum among all sequences that are lexicographically smaller than M, i.e., there does not exist a sequence $N \in \mathcal{M}(n, \Delta) \setminus \{\mathrm{P}(M)\}$ such that $M \succ N \succ \mathrm{P}(M)$ holds.

For a sequence $M \in \mathcal{M}(n, \Delta)$, we next give the structure of the predecessor $\mathrm{P}(M)$, if it exists, of M.

Theorem 3.3. *Let $n \geq 2$ and $\Delta \geq 0$ be two integers and $M = ((a_1, b_1), (a_2, b_2), \ldots, (a_{n-1}, b_{n-1}))$ be a sequence in $\mathcal{M}(n, \Delta)$. Let d denote the root-degree of M. Then, for the predecessor $\mathrm{P}(M) = ((x_1, y_1), (x_2, y_2), \ldots, (x_{n-1}, y_{n-1}))$, if it exists, we have*

(a) *If $a_i = 1$ and $b_i = 0$, $\forall i \in [1, d]$, then $\mathrm{P}(M)$ does not exist.*

(b) *If $a_i \neq 1$ for some $i \in [1, d]$, $b_j = 0$ and $M(j) = \mathrm{S}(a_j, b_j)$, $\forall j \in [1, d]$. Then, for the largest integer $k \in [1, d]$ such that $a_k \neq 1$, it holds that $\mathrm{d}(\mathrm{P}(M)) = k - 1 + \lceil (a_k + d - k)/(a_k - 1) \rceil$, $y_1 = \Delta$, $y_i = 0, \forall i \in [2, \mathrm{d}(\mathrm{P}(M))]$, $x_i = a_i, \forall i \in [1, k-1]$, $x_i = a_k - 1, \forall i \in [k, \mathrm{d}(\mathrm{P}(M)) - 1]$, $x_{\mathrm{d}(\mathrm{P}(M))} = a_k + d - k - \lfloor (a_k + d - k)/(a_k - 1) \rfloor (a_k - 1)$, and $\mathrm{P}(M)(i) = \mathrm{L}(x_i, y_i)$, $\forall i \in [1, \mathrm{d}(\mathrm{P}(M))]$.*

(c) *If $b_i \neq 0$ for some $i \in [1, d]$, $M(j) = \mathrm{S}(a_j, b_j)$, $\forall j \in [1, d]$. For the largest integer $k \in [1, d]$ such that $b_k \neq 0$, let $p \triangleq \max\{i \leq d \mid a_k = a_{k+i}\}$, $q \triangleq p + 1$ if $b_k = 1$, and $q \triangleq \lfloor (\Delta - \sum_{1 \leq i \leq k-1} b_i)/(b_k - 1) \rfloor$ if $b_k \geq 2$ and $t \triangleq \min\{q, p+1\}$. Then, it holds that $\mathrm{d}(\mathrm{P}(M)) = d$, $x_i = a_i, \forall i \in [1, d]$, $y_i = b_i, \forall i \in [1, k-1]$, $y_k = b_k - 1$; for $k \neq d$, we have $y_{k+i} = b_k - 1, \forall i \in [1, t-1]$, $y_{k+t} = (\Delta - \sum_{1 \leq i \leq k-1} b_i) - t(b_k - 1)$, $y_i = 0, \forall i \in [k + t + 1, d]$, and $\mathrm{P}(M)(i) = \mathrm{L}(x_i, y_i)$, $\forall i \in [1, d]$.*

(d) *Otherwise if $a_i \neq 1, b_j \neq 0$, $M(\ell) \neq \mathrm{S}(a_\ell, b_\ell)$, for some $i, j, \ell \in [1, d]$. For the largest integer $k \in [1, d]$ such that $M(k) \neq \mathrm{S}(a_k, b_k)$, let $p \triangleq \max\{i \leq d \mid a_k = a_{k+i}$ and $b_k = b_{k+1}\}$. Then, it holds that $\mathrm{d}(\mathrm{P}(M)) = d$, $(x_i, y_i) = (a_i, b_i), \forall i \in [1, d]$, $\mathrm{P}(M)(i) = M(i)$, $\forall i \in [1, k-1]$, $\mathrm{P}(M)(k+i) = \mathrm{P}(M(k))$, $\forall i \in [0, p]$, and $\mathrm{P}(M)(i) = \mathrm{L}(a_i, b_i)$, $\forall i \in [k + p + 1, d]$.*

Proof. (a) If $a_i = 1$ and $b_i = 0$, $\forall i \in [1, d]$, then it holds that $d = n - 1$. Thus, by Lemma 3.10(i), it holds that $M = \mathrm{S}(n, \Delta)$, and therefore, $\mathrm{P}(M)$ does not exist.

(b) If $a_i \neq 1$ for some $i \in [1, d]$, $b_j = 0$ and $M(j) = \mathrm{S}(a_j, b_j)$, $\forall j \in [1, d]$, then M is lexicographically minimum among all those sequences $((c_1, c'_1), \ldots, (c_{n-1}, c'_{n-1})) \in \mathcal{M}(n, \Delta)$ for which it holds that $c_i = a_i, \forall i \in [1, d]$. Further, $a_i \neq 1$ for some $i \in [1, d]$ implies that $\mathrm{P}(M)$ exists. By the definition of a predecessor, observe that $\mathrm{P}(M)$ is lexicographically maximum among all those sequences

$S = ((s_1, s'_1), \ldots, (s_{n-1}, s'_{n-1})) \in \mathcal{M}(n, \Delta)$ for which it holds that $s_i = x_i, \forall i \in [1, \mathrm{d}(\mathrm{P}(M))]$. This implies that for each such sequence S, it holds that either $(y_1, y_2, \ldots, y_{\mathrm{d}(\mathrm{P}(M))}) \succ (s'_1, s'_2, \ldots, s'_{\mathrm{d}(\mathrm{P}(M))})$ or $y_i = s'_i$ and $\mathrm{P}(M)(i) \succeq S(i), \forall i \in [1, \mathrm{d}(\mathrm{P}(M))]$. The former implies that $y_1 = \Delta$, $y_i = 0, \forall i \in [2, \mathrm{d}(\mathrm{P}(M))]$, while the latter implies that $\mathrm{P}(M)(i) = \mathrm{L}(x_i, y_i)$, $\forall i \in [1, \mathrm{d}(\mathrm{P}(M))]$. Further, the sequence $(x_1, \ldots, x_{\mathrm{d}(\mathrm{P}(M))})$ satisfies Lemma 3.9(i), and there does not exist a sequence L that satisfies Lemma 3.9(i) such that $(x_1, \ldots, x_{\mathrm{d}(\mathrm{P}(M))}) \prec L \prec (a_1, \ldots, a_d)$ holds by the definition of $\mathrm{P}(M)$. This and the definition of k imply that $x_i = a_i, \forall i \in [1, k-1]$; the sequence $(x_k, x_{k+1}, \ldots, x_{\mathrm{d}(\mathrm{P}(M))})$ is a non-decreasing sequence, and there there does not exist a sequence $Z = (z_1, \ldots, z_t)$ such that $z_i \in [1, a_k - 1], \forall i \in [1, t]$ and $\sum_{1 \le i \le t} z_i = \sum_{k \le j \le d} a_j$ for which it holds that $(x_k, \ldots, x_{\mathrm{d}(\mathrm{P}(M))}) \prec Z \prec (a_k, \ldots, a_d)$ holds. This eventually implies that $\mathrm{d}(\mathrm{P}(M)) = k - 1 + \lceil \sum_{k \le i \le d} a_i/(a_k - 1) \rceil$, $x_i = a_k - 1, \forall i \in [k, \mathrm{d}(\mathrm{P}(M)) - 1]$, and $x_{\mathrm{d}(\mathrm{P}(M))} = \sum_{k \le i \le d} a_i - \lfloor \sum_{k \le i \le d} a_i/(a_k - 1) \rfloor (a_k - 1)$. Thus, by the definition of k, we have $\sum_{k \le i \le d} a_i = a_k + d - k$, and therefore, we have the required result. (c) If $b_i \neq 0$ for some $i \in [1, d]$ and $M(j) = \mathrm{S}(a_j, b_j)$, $\forall j \in [1, d]$, then M is lexicographically minimum among all those sequences $((c_1, c'_1), \ldots, (c_{n-1}, c'_{n-1})) \in \mathcal{M}(n, \Delta)$ for which it holds that $(c_i, c'_i) = (a_i, b_i), \forall i \in [1, d]$. Since $b_i \neq 0$ for some $i \in [1, d]$, therefore $\mathrm{P}(M)$ exists and is lexicographically maximum among all those sequences $((s_1, s'_1), \ldots, (s_{n-1}, s'_{n-1})) \in \mathcal{M}(n, \Delta)$ for which it holds that $(s_i, s'_i) = (a_i, y_i), \forall i \in [1, \mathrm{d}(\mathrm{P}(M))]$. This implies that $x_i = a_i, \forall i \in [1, \mathrm{d}(\mathrm{P}(M))]$, and therefore, we have $\mathrm{d}(\mathrm{P}(M)) = d$. Furthermore, $(y_1, y_2, \ldots, y_d) \succ (s'_1, s'_2, \ldots, s'_d)$ and $\mathrm{P}(M)(i) \succeq S(i), \forall i \in [1, d]$. This implies that for each integer $i \in [1, d]$, it holds that $\mathrm{P}(M)(i) = \mathrm{L}(a_i, y_i)$, and the sequence $(y_1, y_2, \ldots, y_d)$ is lexicographically minimum among all those sequences that satisfy Lemma 3.9(ii) and are lexicographically smaller than $(b_1, b_2, \ldots, b_d)$. This and the definition of k imply that $y_i = b_i, \forall i \in [1, k-1]$; by Lemma 3.9(ii), the sequence $(y_k, y_{k+1}, \ldots, y_{k+p})$ is a non-decreasing sequence, and there does not exist a sequence $Z = (z_k, z_{k+1}, \ldots, z_{k+p})$ such that $z_i \in [0, b_k - 1]$ and $\sum_{k \le i \le k+p} z_i = \min\{(p+1)(b_k - 1), \Delta - \sum_{1 \le i \le k-1} b_i\}$ for which it holds that $(y_k, y_{k+1} \ldots, x_{k+p}) \prec Z \prec (b_k, b_{k+1}, \ldots, b_{k+p})$. This eventually implies that $y_k = b_k - 1$ and for $k \neq d$ $y_{k+i} = b_k - 1, \forall i \in [1, t-1]$. This and the minimality of $(y_1, y_2, \ldots, y_d)$ imply that $y_{k+t} = (\Delta - \sum_{1 \le i \le k-1} b_i) - t(b_k - 1)$ and $y_i = 0, \forall i \in [k + t + 1, d]$.

(d) The conditions of this case imply that M is not the lexicographically minimum among all those sequences $((c_1, c'_1), \ldots, (c_{n-1}, c'_{n-1})) \in \mathcal{M}(n, \Delta)$ for which it holds that $(c_i, c'_i) = (a_i, b_i), \forall i \in [1, d]$. This implies that for $\mathrm{P}(M)$, it holds that $\mathrm{d}(\mathrm{P}(M)) = d$ and $(x_i, y_i) = (a_i, b_i), \forall i \in [1, d]$, and there does not exist a sequence $S = ((s_1, s'_1), \ldots, (s_{n-1}, s'_{n-1})) \in \mathcal{M}(n, \Delta)$ for which it holds that

$(s_i, s'_i) = (a_i, b_i), \forall i \in [1, d]$ and $((x_1, y_1), \ldots, (x_{n-1}, y_{n-1})) \prec S \prec M$. This implies that for each such sequence S, it holds that $\mathrm{P}(M)(1) \oplus \mathrm{P}(M)(2) \oplus \cdots \oplus \mathrm{P}(M)(d) \succ S(1) \oplus S(2) \oplus \cdots \oplus S(d)$. This and the definition of k imply that $\mathrm{P}(M)(i) = M(i)$, $\forall i \in [1, k-1]$. Furthermore, by Lemma 3.9(iii) and the definition of p, it holds that $\mathrm{P}(M)(i) \succ \mathrm{P}(M)(i+1)$, $\forall i \in [k, k+p-1]$, and there does not exist a sequence $S = ((s_1, s'_1), \ldots, (s_{n-1}, s'_{n-1})) \in \mathcal{M}(n, \Delta)$ such that $(s_i, s'_i) = (a_i, b_i), \forall i \in [1, d]$ and $S(i) \succ S(i+1), \forall i \in [k, k+p-1]$ for which it holds that $\mathrm{P}(M)(k) \oplus \mathrm{P}(M)(k+1) \oplus \cdots \oplus \mathrm{P}(M)(K+1) \prec S(k) \oplus S(k+1) \oplus \cdots \oplus S(k+p+1) \prec M(k) \oplus M(k+1) \oplus \cdots \oplus M(k+p+1)$. Since $M(k) \neq \mathrm{S}(a_k, b_k)$, therefore $\mathrm{P}(M)(k)$ is a lexicographically minimum sequence in $\mathcal{M}(a_k, b_k)$ for which it holds that $\mathrm{P}(M)(k) \prec M(k)$. This implies that $\mathrm{P}(M)(k+i) = \mathrm{P}(M(k)), \forall i \in [0, p]$. Further, by the minimality of $\mathrm{P}(M)$, $\mathrm{P}(M)(i) = \mathrm{L}(a_i, b_i)$, $\forall i \in [k+p+1, d]$.

Finally, one can easily verify that the sequence $((x_1, y_1), \ldots, (x_{n-1}, y_{n-1}))$ obtained in each of the above cases satisfies Lemma 3.9(i)–(iii) by construction, and hence, $((x_1, y_1), \ldots, (x_{n-1}, y_{n-1}))$ is an element of $\mathcal{M}(n, \Delta)$ that is $\mathrm{P}(M)$, which completes the proof. $\square$

Lemma 3.11. *Let $n \geq 2$ and $\Delta \geq 0$ be two integers and M be a sequence in $\mathcal{M}(n, \Delta)$. Then, the predecessor $\mathrm{P}(M)$, if it exits, can be computed in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

The proof of Lemma 3.11 follows from Algorithm 2 and Lemma 3.12.

We next present Algorithm 2 to compute the predecessor based on Theorem 3.3. In this algorithm, for a sequence $M \in \mathcal{M}(n, \Delta)$ with root-degree d and integer $i \in [1, d]$, the variable $M[i]$ stores the i-th root-subsequence $M(i)$ of M, and the variable $\mathrm{P}[M]$ stores the predecessor, if it exists, of M.

Algorithm 2 Computing the Predecessor of an Admissible Sequence

```
Input: Two integers $n \geq 2$ and $\Delta \geq 0$ and an $(n, \Delta)$-admissible sequence
    $M = ((a_1, b_1), \ldots, (a_{n-1}, b_{n-1}))$.
Output: The predecessor of $M$ if it exists; The predecessor of $M$ does not
    exist otherwise.
1: $d :=$ The root-degree of $M$;
2: if If $a_i = 1$ and $b_i = 0, \forall i \in [1, d]$ then
3:   Output The predecessor of $M$ does not exist   /* Theorem 3.3(a) */
4: else/* The predecessor of $M$ exists by Theorem 3.3*/
5:   $\mathrm{P}[M] := ((x_1, y_1), \ldots, (x_{n-1}, y_{n-1}))$
6:   if $a_i \neq 1$ for some $i \in [1, d]$, $b_j = 0$ and $M[j] = \mathrm{S}(a_j, b_j)$, $\forall j \in [1, d]$ then
       /* Theorem 3.3(b) */
7:     $k := \max\{i \mid a_i \neq 1\}$; $h := k - 1 + \lceil (a_k + d - k)/(a_k - 1) \rceil$; /* The root-degree
```

```
          d($P(M)$) of $P(M)$ */
 8:     $y_1 := \Delta$; $y_i := 0, \forall i \in [2,h]$; $x_i := a_i, \forall i \in [1,k-1]$;
          $x_i := a_k-1, \forall i \in [k,h-1]$; $x_h := a_k+d-k-\lfloor (a_k+d-k)/(a_k-1)\rfloor (a_k-1)$;
          P[$M$][$i$] = L($x_i, y_i$), $\forall i \in [1,h]$
 9:   else if $b_i \neq 0$ for some $i \in [1,d]$ and $M[j] = \mathrm{S}(a_j,b_j)$, $\forall j \in [1,d]$ then
        /* Theorem 3.3(c) */
10:     $k := \max\{i \mid b_i \neq 0\}$; $p := \max\{i \leq d \mid a_k = a_{k+i}\}$; $q := p+1$ if $b_k = 1$;
          $q := \lfloor (\Delta - \sum_{1\leq i\leq k-1} b_i)/(b_k-1)\rfloor$ if $b_k \geq 2$; $t := \min\{q, p+1\}$;
11:     $x_i := a_i, \forall i \in [1,d]$; $y_i := b_i, \forall i \in [1,k-1]$; $y_k := b_k - 1$;
12:     if $k \neq d$ then
13:       $y_{k+i} := b_k - 1, \forall i \in [1,t-1]$; $y_{k+t} := (\Delta - \sum_{1\leq i\leq k-1} b_i) - t(b_k-1)$;
          $y_i = 0, \forall i \in [k+t+1,d]$
14:     end if;
15:     P[$M$][$i$] := L($x_i, y_i$), $\forall i \in [1,d]$
16:   else/* If $a_i \neq 1, b_j \neq 0$ and $M[\ell] \neq \mathrm{S}(a_\ell,b_\ell)$, for some $i,j,\ell \in [1,d]$ */
        /* Theorem 3.3(d) */
17:     $k := \max\{i \mid M[i] \neq \mathrm{S}(a_i,b_i)\}$;
18:     $p := \max\{i \leq d \mid a_k = a_{k+i}$ and $b_k = b_{k+1}\}$;
          $(x_i,y_i) := (a_i,b_i), \forall i \in [1,d]$; P[$M$][$i$] := $M[i]$, $\forall i \in [1,k-1]$;
19:     P[$M$][$k$] := Algorithm 1($a_k, b_k, M[k]$);
20:     P[$M$][$k+i$] := P[$M$][$k$], $\forall i \in [1,p]$; P[$M$][$i$] := L($a_i,b_i$), $\forall i \in [k+p+1,d]$
21:   end if;
22:   Output P[$M$] as the predecessor of $M$;
23: end if.
```

Lemma 3.12. *For two integers $n \geq 2$ and $\Delta \geq 0$ and an (n,Δ)-admissible sequence M, Algorithm 2 outputs the predecessor* $\mathrm{P}(M)$*, if it exits, in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space.*

Proof. Correctness: The correctness of Algorithm 2 immediately follows from Theorem 3.3.
Complexity analysis: By the definition of the root-degree, we can compute d at Line 1 in $\mathcal{O}(n)$ time.

We can test if $a_i \neq 1$ and $b_j \neq 0$ hold for some $i,j \in [1,d]$ in $\mathcal{O}(n)$ time. Similarly, we can test if $M(j) = \mathrm{S}(a_j,b_j)$, for some $j \in [1,d]$ in $\mathcal{O}(n)$ time, since the length of $M(j)$ is $a_j - 1$, and $\sum_{1\leq i\leq d} a_i = n-1$. Hence, we can test the conditions at Lines 2, 6, 9, and 16 in $\mathcal{O}(n+n) = \mathcal{O}(n)$ time. This implies that we can check if the predecessor of M exists in $\mathcal{O}(n)$ time. We next discuss the time complexity of computing the predecessor in each of the cases at Lines 6, 9, and 16.

When the conditions at Line 6 hold, then k and h can be computed in $\mathcal{O}(n)$ time, since $k \leq d$ and $h \leq d+1$. This implies that $((x_1, y_1), \ldots, (x_h, y_h))$ can be computed in $\mathcal{O}(n)$ time. Furthermore, we can compute $\mathrm{L}(x_i, y_i), \forall i \in [1, h]$ in $\mathcal{O}(x_i)$ time by Lemma 3.10(i). Recall that for $\mathrm{P}[M]$, it holds that $\sum_{1 \leq i \leq d} x_i = n-1$. Thus, $\mathrm{P}[M]$ can be computed in $\mathcal{O}(n)$ time.

When the conditions at Line 9 hold, then k, p, q, and t can be computed in $\mathcal{O}(n)$ time by there definitions. Thus, $((x_1, y_1), \ldots, (x_d, y_d))$ can be computed from Line 11 to Line 14 in $\mathcal{O}(n)$ time. Furthermore, $\mathrm{L}(x_i, y_i), \forall i \in [1, d]$ can be computed at Line 15 in $\mathcal{O}(n)$ time, as discussed above. This implies that $\mathrm{P}[M]$ can be computed in $\mathcal{O}(n)$ time in this case.

Finally, when the conditions at Line 16 hold, then once again, we can compute k in $\mathcal{O}(n)$ time, since $\sum_{1 \leq i \leq d} a_i = n-1$. Further, p can be computed in $\mathcal{O}(n)$ time by the definition of p. However, $\mathrm{P}[M][k]$ can be obtained by recursively running Algorithm 1 on a_k, b_k and $M[k]$. Note that this operation is repeated at most the length of the sequence $\mathrm{P}[M[k]]$, which is $a_k - 1$ in this case, and hence, $\mathrm{P}[M][k]$ can be computed in $\mathcal{O}(n)$ time. Once again, the computation at Line 20 can be done in $\mathcal{O}(n)$ time since it holds that $\sum_{1 \leq i \leq d} a_i = n-1$. Hence, $\mathrm{P}[M]$ can be computed in $\mathcal{O}(n)$ time .

Observe that the $\mathcal{O}(n)$ space is sufficient to store $\mathrm{P}[M]$, if it exists, which completes the proof. □

Note that we can generate all rooted graphs in $\mathcal{H}(n, \Delta)$ by generating their canonical representation by repeatedly using Algorithm 2 starting from the lexicographically maximum sequence $\mathrm{L}(n, \Delta)$ in $\mathcal{M}(n, \Delta)$ in $\mathcal{O}(n)$ time per graph and $\mathcal{O}(n)$ space in total.

Theorem 3.4. *Let $n \geq 2$ and $\Delta \geq 0$ be two integers. Then, all mutually non-isomorphic graphs with n vertices, Δ self-loops, and a tree skeleton can be generated in $\mathcal{O}(n)$ time per graph and $\mathcal{O}(n)$ space in total.*

Proof. A tree can be viewed as a rooted tree by considering its centroid as the root [41]. We know that when n is odd, then there are only trees with unicentroids; however, when n is even, then there are trees with unicentroids.

By the definition of a unicentroid, all mutually non-isomorphic graphs with n vertices, Δ self-loops, and a tree skeleton with a unicentroid can be enumerated by generating all graphs H in $\mathcal{H}(n, \Delta)$ such that each root subgraph of H has at most $\lfloor (n-1)/2 \rfloor$ number of vertices, i.e., by generating all sequences $M = ((a_1, b_1), \ldots, (a_{n-1}, b_{n-1})) \in \mathcal{M}(n, \Delta)$ such that $a_i \in [1, \lfloor (n-1)/2 \rfloor]$, $\forall i \in [1, \mathrm{d}(\mathrm{P}(M))]$. Let S denote the sequence that is lexicographically maximum among all those sequences in $\mathcal{M}(n, \Delta)$ that represent a tree with a unicentroid.

When n is even, then it holds that $S = ((\lfloor (n-1)/2 \rfloor, \Delta), (\lfloor (n-1)/2 \rfloor, 0), (1, 0)) \oplus \mathrm{L}(\lfloor (n-1)/2 \rfloor, \Delta) \oplus \mathrm{L}(\lfloor (n-1)/2 \rfloor, 0) \oplus \mathrm{L}(1, 0)$. Recall that $\mathrm{L}(1, 0)$ is an empty sequence, and therefore, we have $S = ((\lfloor (n-1)/2 \rfloor, \Delta), (\lfloor (n-1)/2 \rfloor, 0), (1, 0)) \oplus \mathrm{L}(\lfloor (n-1)/2 \rfloor, \Delta) \oplus \mathrm{L}(\lfloor (n-1)/2 \rfloor, 0)$. However, when n is odd, then it holds that $S = ((\lfloor (n-1)/2 \rfloor, \Delta), (\lfloor (n-1)/2 \rfloor, 0)) \oplus \mathrm{L}(\lfloor (n-1)/2 \rfloor, \Delta) \oplus \mathrm{L}(\lfloor (n-1)/2 \rfloor, 0)$. Hence, we can generate all sequences in $\mathcal{M}(n, \Delta)$ that represent a graph $H \in \mathcal{H}(n, \Delta)$ such that the skeleton of H has a unicentroid by repeatedly using Theorem 3.3 starting from the sequence S. This implies that we can generate all such sequences in $\mathcal{O}(n)$ time per sequence and $\mathcal{O}(n)$ space by using Algorithm 2.

When n is even, then all mutually non-isomorphic graphs with n vertices, Δ self-loops, and a tree skeleton with a bicentroid can be enumerated by generating all sequences $M = ((a_1, b_1), \ldots, (a_{n-1}, b_{n-1})) \in \mathcal{M}(n, \Delta)$ such that the root-degree $\mathrm{d}(M) = 2$, $a_1 = a_2 = n/2$ and $b_1 + b_2 = \Delta$ with $b_1 \geq b_2$. In such a case, the sequences $((n/2, \Delta), (n/2, 0)) \oplus \mathrm{L}(n/2, \Delta) \oplus \mathrm{L}(n/2, 0)$ and $((n/2, \lceil \Delta/2 \rceil), (n/2, \lfloor \Delta/2 \rfloor)) \oplus \mathrm{L}(n/2, \lceil \Delta/2 \rceil) \oplus \mathrm{L}(n/2, \lfloor \Delta/2 \rfloor)$ are lexicographically maximum and minimum, respectively, among all those sequences in $\mathcal{M}(n, \Delta)$ that represent a graph with a bicentroid. Let $M = ((a_1, b_1), \ldots, (a_{n-1}, b_{n-1})) \in \mathcal{M}(n, \Delta)$ be a sequence that represents a graph $H \in \mathcal{H}(n, \Delta)$ such that the skeleton of H has a bicentroid and $M \neq ((n/2, \lceil \Delta/2 \rceil), (n/2, \lfloor \Delta/2 \rfloor)) \oplus \mathrm{L}(n/2, \lceil \Delta/2 \rceil) \oplus \mathrm{L}(n/2, \lfloor \Delta/2 \rfloor)$. When $M(i) = \mathrm{S}(a_j, b_j)$, $\forall i \in \{1, 2\}$, then it holds that $\mathrm{P}(M) = ((a_1, b_1 - 1), (a_2, b_2 + 1)) \oplus \mathrm{L}(a_1, b_1 - 1) \oplus (a_2, b_2 + 1)$. However, in the case otherwise, i.e., when $M(i) \neq \mathrm{S}(a_j, b_j)$, for some $i \in \{1, 2\}$, then $\mathrm{P}(M)$ can be generated by using Theorem 3.3(d). Clearly, in both of these cases, $\mathrm{P}(M)$ can be generated in $\mathcal{O}(n)$ time and $\mathcal{O}(n)$ space. This eventually implies that all sequences that represent a graph in $\mathcal{H}(n, \Delta)$ with a bicentroid can be generated in $\mathcal{O}(n)$ time per sequence and $\mathcal{O}(n)$ space by repeatedly computing the predecessor of sequences M as described above starting from $((n/2, \Delta), (n/2, 0)) \oplus \mathrm{L}(n/2, \Delta) \oplus \mathrm{L}(n/2, 0)$ until $M = ((n/2, \lceil \Delta/2 \rceil), (n/2, \lfloor \Delta/2 \rfloor)) \oplus \mathrm{L}(n/2, \lceil \Delta/2 \rceil) \oplus \mathrm{L}(n/2, \lfloor \Delta/2 \rfloor)$.

Hence, we can generate all non-isomorphic graphs with n vertices, Δ self-loops, and a tree skeleton with a unicentroid or bicentroid in $\mathcal{O}(n)$ time per graph and $\mathcal{O}(n)$ space in total, which completes that proof. □

3.4 Results and Application

We computed all graphs with n vertices, Δ self-loops, and a tree skeleton for different values of n and Δ to test the efficiency of our algorithm, and the results are listed in Table 3.1. These experiments were performed on a PC with an Intel Core i7-500 processor, running at 2.70 GHz, 16 GB of memory, and Windows 10. From the experimental results, it is evident that the proposed method is computationally efficient.

Table 3.1. Experimental results of the enumeration method.

(n, Δ)	# Generated Graphs	Time (sec.)
(5, 9)	856	0.278
(5, 10)	1186	0.213
(5, 30)	50,596	4.354
(6, 9)	4270	0.992
(6, 10)	6333	1.571
(6, 30)	619,431	141.334
(7, 9)	20,548	5.084
(7, 10)	32,337	7.047
(8, 9)	95,357	17.444
(8, 10)	159,058	31.755
(9, 9)	429,496	88.899
(9, 10)	756,045	185.823
(10, 9)	1,882,764	528.286
(10, 10)	3,488,567	914.806
(17, 2)	25,939,679	3911.33
(18, 0)	123,867	34.189
(20, 0)	823,065	334.357
(22, 0)	5,623,756	1807.53
(24, 0)	39,299,897	8042.88

Observe that a graph with a tree skeleton, Δ self-loops, and no multiple edges has cycle rank Δ. Therefore, the class of such graphs contains all polymer topologies of cycle rank Δ with a tree skeleton. However, it is a natural question to search for a relationship between the number n of vertices and the number Δ of self-loops such that there exists a polymer topology with a tree skeleton, n vertices, and cycle rank Δ. Clearly, for $n = 1$ and $\Delta \geq 2$, there exists exactly one polymer topology with a tree skeleton, n vertices, and Δ self-loops. Let $\mathcal{P}(n, \Delta)$ denote a maximal set of mutually non-isomorphic polymer topologies with a tree skeleton, n vertices, Δ self-loops, and no multiple edges. Recall that, we proved a necessary condition on Δ to have a polymer topology with a tree skeleton and n vertices in Lemma 2.7. By this lemma, if $n \geq 1$ and $\Delta \geq \lceil \frac{n}{2} \rceil + 1$, then it holds that $\mathcal{P}(n, \Delta) \neq \emptyset$. Let $\mathcal{G}(n, \Delta)$ denote a maximal set of mutually non-isomorphic graphs with a tree skeleton, n vertices, and Δ self-loops. For an integer $r \geq 1$, let $\mathcal{P}(r)$ denote a maximal set of mutually non-isomorphic polymer topologies with a tree skeleton, n vertices, and r self-loops. By Lemma 2.7, it holds that

$$\mathcal{P}(r) = \bigcup_{n \in \mathbb{Z}^+ : \lceil \frac{n}{2} \rceil + 1 \leq r} \mathcal{P}(n,r) \subseteq \bigcup_{n \in \mathbb{Z}^+ : \lceil \frac{n}{2} \rceil + 1 \leq r} \mathcal{G}(n,r) \tag{3.4.2}$$

Thus, by using Equation (3.4.2) and identifying the degree of each vertex in the graphs in $\mathcal{G}(n,r)$ from their canonical representations, we can compute all polymer topologies in $\mathcal{P}(r)$. We applied our method to generate all polymer topologies in $\mathcal{P}(r)$ for rank $r = 2, 3, \ldots, 9$, and the results are listed in Table 3.2.

Table 3.2. The number of polymer topologies with a tree skeleton, r self-loops, and no multiple edges.

	Rank r							
n	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1
2	1	1	2	2	3	3	4	4
3	–	1	2	4	6	9	12	16
4	–	1	3	6	13	21	35	51
5	–	–	2	7	18	40	77	136
6	–	–	1	6	23	61	147	300
7	–	–	–	3	20	76	223	559
8	–	–	–	1	14	74	288	868
9	–	–	–	–	5	54	291	1128
10	–	–	–	–	2	29	241	1212
11	–	–	–	–	–	10	145	1057
12	–	–	–	–	–	2	68	733
13	–	–	–	–	–	–	19	390
14	–	–	–	–	–	–	4	151
15	–	–	–	–	–	–	–	38
16	–	–	–	–	–	–	–	6
Total	2	4	11	30	105	308	1555	6650

3.5 Concluding Remarks

We proposed an efficient method to enumerate all mutually non-isomorphic graphs with a tree skeleton, a given number of vertices, and the number of self-loops.

The idea of this method is to generate rooted graphs with n vertices and Δ self-loops by generating their canonical representation. We defined the canonical representation of a rooted graphs H with n vertices and Δ self-loops based on the ordered graphs of H. The proposed method generates all graphs with a tree skeleton, n vertices, and Δ self-loops in $\mathcal{O}(n)$ time per tree and $\mathcal{O}(n)$ space in total. As an application, we can generate all polymer topologies with a tree skeleton, self-loops, no multiple edges, and a given cycle rank.

An interesting future research direction is to design a method that can directly count and enumerate all mutually non-isomorphic polymer topologies with a given cycle rank.

4 Some Theoretical Results Related to Pairwise Compatibility Graphs

4.1 Introduction

The pairwise compatibility tree construction problem (PCTCP) asks to find evidence to show if the given graph G is a PCG or not. When a graph G is a PCG, then there exists at least one tuple $(T, w, d_{\min}, d_{\max})$ such that $G = \text{PCG}(T, w, d_{\min}, d_{\max})$, in which case the tuple will be an evidence for G to be PCG. Thus, to show if a graph G is not a PCG, we need to prove that there does not exist a tuple $(T, w, d_{\min}, d_{\max})$ such that $G = \text{PCG}(T, w, d_{\min}, d_{\max})$, where there is an infinite search space of an edge weight function w and two reals $d_{\min}$ and $d_{\max}$. Therefore it is a challenging task to construct an evidence with a finite size to verify that G is not a PCG.

The aim of this chapter is three-fold. First, we proposed two LP formulations to solve the PCTCP. To see if a given graph G with n vertices is a PCG or not, we try to construct a tuple $(T, w, d_{\min}, d_{\max})$ such that $G = \text{PCG}(T, w, d_{\min}, d_{\max})$. For this, we fix a tuple (G, T, σ, λ) of a tree T, a bijection σ between the vertices of G and the leaves of T, and a coloring λ on the set of non-adjacent vertex pairs in G to a color set $\{0, 1\}$, where each non-adjacent vertex pair $\{u, v\}$ in G colored with $\lambda(uv) = 0$ (resp., $\lambda(uv) = 1$) is restricted to have a distance between the corresponding leaves $\sigma(u)$ and $\sigma(v)$ in T smaller than $d_{\min}$ (resp., larger than $d_{\max}$). For each tuple (G, T, σ, λ), we formulate a linear programming (LP) formulation, $\text{LP}(G, T, \sigma, \lambda)$ with $m + 2$ variables, where m is the number of edges in T, and $\mathcal{O}(n^2)$ constraints such that $\text{LP}(G, T, \sigma, \lambda)$ is feasible if and only if:

(i) There exists a weight function w and two real numbers $d_{\min}, d_{\max} \in \mathbb{R}_+$ such that for each edge uv in G it holds that the distance between $\sigma(u)$ and $\sigma(v)$ in T is in the closed interval $[d_{\min}, d_{\max}]$; and

(ii) For each pair u, v of non-adjacent vertices in G it holds that the distance between $\sigma(u)$ and $\sigma(v)$ in T is strictly less than $d_{\min}$ if $\lambda(uv) = 0$ and strictly greater than $d_{\max}$ otherwise ($\lambda(uv) = 1$).

Instead of searching for infinitely many reals w, $d_{\min}$, and $d_{\max}$ until $G = \mathrm{PCG}(T, w, d_{\min}, d_{\max})$ holds, it suffices to solve this single $\mathrm{LP}(G, T, \sigma, \lambda)$. Note that there is no such set of reals w, $d_{\min}$, and $d_{\max}$ if $\mathrm{LP}(G, T, \sigma, \lambda)$ is infeasible. However, due to numerical errors, the infeasibility of $\mathrm{LP}(G, T, \sigma, \lambda)$ cannot be directly employed as an evidence for NPCGs. For this, we propose another LP formulation, $\mathrm{DLP}(G, T, \sigma, \lambda)$ with $\mathcal{O}(n^2)$ variables and $m+3$ constraints such that $\mathrm{DLP}(G, T, \sigma, \lambda)$ is feasible if and only if the $\mathrm{LP}(G, T, \sigma, \lambda)$ is infeasible. Thus the solution to a feasible $\mathrm{DLP}(G, T, \sigma, \lambda)$ can play as a role of an evidence of NPCGs.

Second, we propose a sufficient condition for a graph G to be a PCG of a given witness tree T by using a new integer linear programming (ILP) formulation.

Third, we study and characterize all tuples (G, T, σ, λ) for $n = 4$ for which $\mathrm{LP}(G, T, \sigma, \lambda)$ is infeasible. Based on these theoretical results, we propose a method to enumerate PCGs with a given number of vertices in Chapter 5.

The rest of the chapter is organized as follows: Section 4.2 reviews some notions related to graph theory and results related to PCGs. Section 4.3 describes the LP formulations to solve the PCTCP. Section 4.4 explains a sufficient condition for a graph to be PCG. We characterize tuples (G, T, σ, λ) for $n = 4$ in Section 4.5. Section 4.6 makes some concluding remarks.

4.2 Preliminaries

This section introduces some basic notions related to graph theory and reviews some known results on PCGs.

4.2.1 Basic Notions

Let $\mathbb{R}$ and $\mathbb{R}_+$ denote the sets of all real numbers and non-negative real numbers, respectively.

Let A be a non-empty finite set and h be an integer with $0 \leq h \leq |A|$. Let $\binom{A}{h}$ denote the family of all subsets $S \subseteq A$ with $|S| = h$. For a given subset $S \subseteq A$ and a function $f : A \to \mathbb{R}$, let $f|_S : S \to \mathbb{R}$ denote the restriction of f on S and $\mathrm{R}(f, S)$ denote the range of the function f over S, i.e., $\mathrm{R}(f, S) = \{f(s) \mid s \in S\}$. For a non-negative integer $k \leq |A| - 1$ and a set K of k distinct integers, a function $\lambda : A \to K$ is called a *k-coloring* of A, and let $\Lambda_k(A)$ denote the set of all k-colorings of A.

Throughout Chapter 4 and 5, the term *graph* stands for a simple undirected graph. Let G be a graph. An edge joining two vertices u and v in G is denoted by uv or vu, where $uv = vu$. The vertex and edge sets of G are denoted by $V(G)$ and $E(G)$, respectively, and the set of all unordered pairs of non-adjacent vertices in

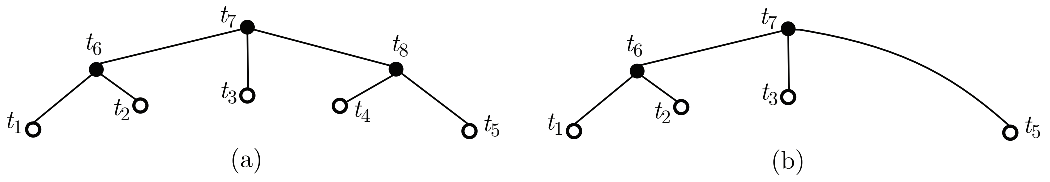


Figure 4.1. An example of tree contraction: (a) A tree T with $V(T) = \{t_i \mid 1 \le i \le 8\}$ and $E(T) = \{t_1t_6, t_2t_6, t_3t_7, t_4t_8, t_5t_8, t_6t_7, t_7t_8\}$; and (b) The tree contraction $T\langle X\rangle$ of T due to $X = \{t_1, t_2, t_3, t_5\}$.

G is denoted by $\overline{E}(G)$. For a vertex v in G, a vertex in G adjacent to v is called a *neighbor* of v in G, the set of all neighbors of v in G is denoted by $N_G(v)$, and the *degree* $\deg_G(v)$ of v in G is defined to be $|N_G(v)|$. A pair $\{u, v\} \in \overline{E}(G)$ is called *false twins* in G if $N_G(v) = N_G(u)$. For a subset $X \subseteq V(G)$, the *subgraph* $G[X]$ *induced by* X is defined to be the subgraph H of G such that $V(H) = X$ and $E(H) = \{uv \in E(G) \mid u, v \in X\}$.

Let T be a tree. A vertex of degree one in T is called a *leaf* in T, and let $L(T)$ denote the set of all leaves in T. For a subset $X \subseteq L(T)$, we define the *tree contraction* $T\langle X\rangle$ of T due to X to be the tree T' such that $L(T') = X$ obtained from T by

(i) removing vertices not contained in any path connecting two vertices in X; and

(ii) regarding the two edges incident to each vertex of degree 2 as a single edge.

Fig. 4.1 illustrates an example of tree contraction. Let w be a function $w : E(T) \to \mathbb{R}_+$, where we denote $w(e)$ or $w(uv)$ for each edge $e = uv \in E(T)$ by $w(u, v)$ for notational convenience. For any two vertices $u, v \in V(T)$, let $E_T(u, v)$ denote the set of all edges in the path between them in T, and define the *edge-distance* (resp., *distance*) $\delta_T(u, v)$ (resp., $d_{T,w}(u, v)$) between them to be $|E_T(u, v)|$ (resp., $\sum_{e\in E_T(u,v)} w(e)$). Let $d_{\min}$ and $d_{\max}$ denote non-negative reals. The graph H with $V(H) = L(T)$ and $E(H) = \{uv \mid d_{\min} \le d_{T,w}(u, v) \le d_{\max}\}$ is denoted by $\mathrm{PCG}(T, w, d_{\min}, d_{\max})$. A graph G is called a *pairwise compatibility graph* (PCG for short) if G is isomorphic to $\mathrm{PCG}(T, w, d_{\min}, d_{\max})$ for some tree T, weight $w : L(T) \to \mathbb{R}_+$ and reals $d_{\min}$ and $d_{\max}$, where T is called a *witness tree* of G, and we denote by $G = \mathrm{PCG}(T, w, d_{\min}, d_{\max})$ when G is $\mathrm{PCG}(T, w, d_{\min}, d_{\max})$. We call a NPCG a *minimal non-pairwise compatibility graph* (MNPCG for short) if every proper induced subgraph of it is a PCG. We call a tree *binary* if each non-leaf vertex is of degree 3. Note that any binary tree with n leaves contains exactly $2n - 3$ edges. It is known that a witness of any PCG G can be chosen as a binary tree (see Lemma 4.14 in the next section). For an integer $n \geq 1$, let $\mathcal{T}_n$ denote a maximal set of binary trees with n leaves no two of which are mutually isomorphic. Note that if $G = \mathrm{PCG}(T, w, d_{\min}, d_{\max})$, then we can also choose w, $d_{\min}$ and $d_{\max}$ so that $0 < d_{\min} < d_{\max}$ and $\{0, d_{\min}, d_{\max}\} \cap \{d_{T,w}(u,v) \mid u, v \in L(T), u \neq v\} = \emptyset$.

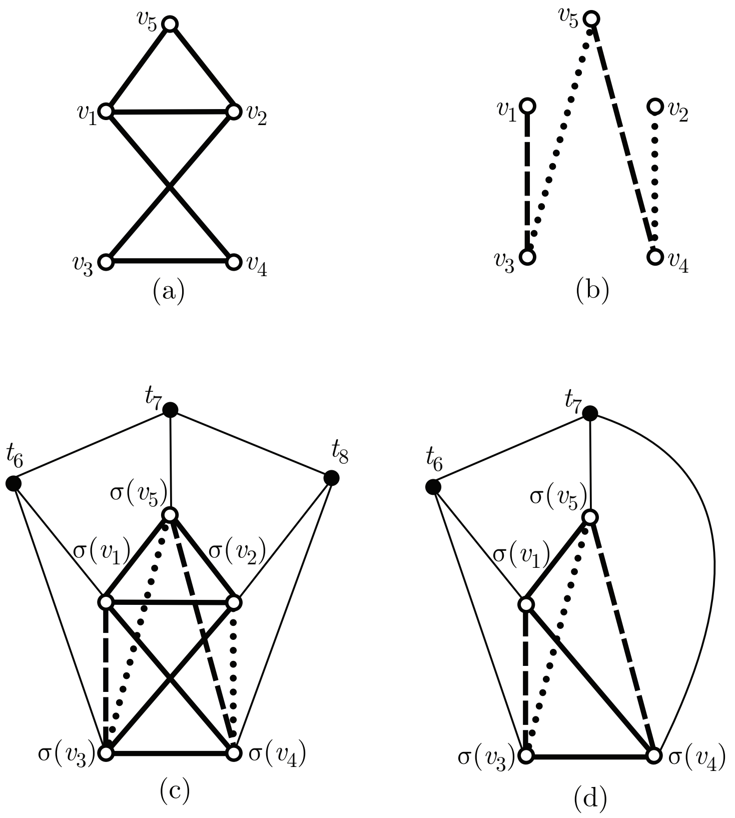


Figure 4.2. Examples of a configuration and a subconfiguration: (a) A graph G with $V(G) = \{v_i \mid 1 \leq i \leq 5\}$ and $E(G) = \{v_1v_2, v_1v_4, v_1v_5, v_2v_3, v_2v_5, v_3v_4\}$ represented by thick solid lines; (b) The set $\overline{E}(G)$ in G with a 2-coloring λ such that $\mathrm{R}(\lambda, \{v_1v_3, v_4v_5\}) = \{0\}$ represented by dashed lines, and $\mathrm{R}(\lambda, \{v_2v_4, v_3v_5\}) = \{1\}$ represented by doted lines; (c) The configuration (G, T, σ, λ), where T is the tree given in Fig. 4.1(a) and σ is defined as $\sigma(v_1) = t_2$, $\sigma(v_2) = t_4$, $\sigma(v_3) = t_1$, $\sigma(v_4) = t_5$, $\sigma(v_5) = t_3$; and (d) The subconfiguration $(G[X], T\langle \mathrm{R}(\sigma, X)\rangle, \sigma|_X, \lambda|_{\overline{E}(G[X])})$ of (G, T, σ, λ) induced by $X = \{v_1, v_3, v_4, v_5\}$.

Let $n \geq 1$ be an integer. Let $\mathcal{G}_n$ denote a maximal set of graphs with n vertices no two of which are mutually isomorphic. Let $G \in \mathcal{G}_n$ and $T \in \mathcal{T}_n$. Let $\Sigma(G, T)$ denote the set of all bijections $\sigma : V(G) \to L(T)$. Recall that $\Lambda_2(\overline{E}(G))$ denotes the set of all 2-colorings $\lambda : \overline{E}(G) \to K$ of $\overline{E}(G)$, where $K = \{0, 1\}$. As observed in the previous section, a bijection $\sigma : V(G) \to L(T)$ determines a correspondence between $V(G)$ and $L(T)$ and a 2-coloring $\lambda \in \Lambda_2(\overline{E}(G))$ introduces a restriction such that $d_{T,w}(\sigma(u), \sigma(v)) < d_{\min}$ (resp., $d_{T,w}(\sigma(u), \sigma(v)) > d_{\max}$) for each pair $uv \in \overline{E}(G)$ with $\lambda(uv) = 0$ (resp., $\lambda(uv) = 1$). We call a tuple (G, T, σ, λ) with $G \in \mathcal{G}_n$, $T \in \mathcal{T}_n$, $\sigma \in \Sigma(G, T)$ and $\lambda \in \Lambda_2(\overline{E}(G))$ a *configuration*. There are $n!2^{|\overline{E}(G)|}$ configurations (G, T, σ, λ) for a given pair of $G \in \mathcal{G}_n$ and $T \in \mathcal{T}_n$. A configuration (G, T, σ, λ) is called *plausible* if $G = \mathrm{PCG}(T, w, d_{\min}, d_{\max})$ for some weight $w : E(T) \to \mathbb{R}_+$, and reals $d_{\min}, d_{\max} \in \mathbb{R}_+$ such that $d_{T,w}(\sigma(u), \sigma(v)) < d_{\min}$ (resp., $d_{T,w}(\sigma(u), \sigma(v)) > d_{\max}$) holds for each pair $uv \in \overline{E}(G)$ with $\lambda(uv) = 0$ (resp., $\lambda(uv) = 1$). Note that a graph G is a PCG if and only if there is a plausible configuration (G, T, σ, λ) for some tree T, bijection σ and 2-coloring λ. A *subconfiguration* of a configuration (G, T, σ, λ) induced by a subset $X \subsetneq V(G)$ is defined to be the configuration $(G[X], T\langle \mathrm{R}(\sigma, X)\rangle, \sigma|_X, \lambda|_{\overline{E}(G[X])})$. Fig. 4.2 illustrates examples of a configuration and a subconfiguration. An implausible configuration with no implausible proper subconfiguration is called a *minimal implausible configuration* (MIC for short). In particular, we denote an MIC with $k \geq 4$ vertices by MICk. Note that every implausible configuration with four vertices is an MIC, since we easily see that all configurations with three vertices are plausible. A configuration (G, T, σ, λ) with no MIC with four vertices is called an *MIC4-free configuration*.

Let $n \geq 1$ be an integer, $G \in \mathcal{G}_n$ and $T \in \mathcal{T}_n$. Two bijections $\sigma_1, \sigma_2 \in \Sigma(G, T)$ are called *equivalent* if there exists an automorphism π of T such that for vertices $u, v, p, q \in V(G)$ with $\sigma_1(u) = \pi(\sigma_2(p))$ and $\sigma_1(v) = \pi(\sigma_2(q))$ it holds that $uv \in E(G)$ if and only if $pq \in E(G)$. Let $\Sigma^*(G, T)$ denote a maximal subset of bijections in $\Sigma(G, T)$ no two of which are mutually equivalent. Next let $\sigma \in \Sigma(G, T)$ and $\lambda_1, \lambda_2 \in \Lambda_2(\overline{E}(G))$. We call 2-colorings λ_1 and λ_2 *equivalent* (with respect to σ) if there exits an automorphism π of T such that for vertices $u, v, p, q \in V(G)$ with $\sigma(u) = \pi(\sigma(p))$ and $\sigma(v) = \pi(\sigma(q))$ it holds that $uv \in \overline{E}(G)$ if and only if $pq \in \overline{E}(G)$ and $\lambda_1(uv) = \lambda_2(pq)$. We call two configurations $(G, T, \sigma, \lambda_1)$ and $(G, T, \sigma, \lambda_2)$ *isomorphic* if λ_1 and λ_2 are equivalent with respect to σ. Let $\Lambda_2^*(G, T, \sigma)$ denote a maximal set of 2-colorings in $\Lambda_2(\overline{E}(G))$ no two of which are equivalent with respect to σ. Observe that for graph G, tree T, any two equivalent bijections $\sigma_1, \sigma_2 \in \Sigma(G, T)$ and 2-coloring $\lambda_1 \in \Lambda_2(\overline{E}(G))$, the configuration $(G, T, \sigma_1, \lambda_1)$ is plausible if and only if there exists a 2-coloring $\lambda_2 \in \Lambda_2(\overline{E}(G))$ such that the configuration $(G, T, \sigma_2, \lambda_2)$ is plausible. Similarly, for any two equivalent colorings $\lambda_1, \lambda_2 \in \Lambda_2(G, T, \sigma)$, the configuration $(G, T, \sigma, \lambda_1)$ is plausible if and only if the configuration $(G, T, \sigma, \lambda_2)$ is plausible. Therefore to verify if G is a PCG due to T it suffices to test the plausibility of configurations (G, T, σ, λ) with $\sigma \in \Sigma^*(G, T)$ and $\lambda \in \Lambda_2^*(G, T, \sigma)$.

4.2.2 Some Known Results on PCGs

It is known that any induced subgraph of a PCG is also a PCG [23]. In other words, any graph that contains an NPCG as a subgraph is not a PCG.

Lemma 4.13. *Any graph that contains an NPCG as an induced subgraph is an NPCG.*

To find a witness tree of a PCG, we only need to search for binary trees based on the next observation reported by Calamoneri et al. [20, 22].

Lemma 4.14. *Every PCG with $n \geq 1$ vertices has a witness tree in the set $\mathcal{T}_n$ of binary trees.*

The next lemma due to Calamoneri et al. [23] states a connectivity property of PCGs.

Lemma 4.15. *A graph is a PCG if each of its connected components is a PCG.*

Xiao and Nagamochi [65] gave a stronger version of Lemma 4.15 by characterizing PCGs via biconnected components.

Lemma 4.16. *A graph is a PCG if and only if each of its biconnected components is a PCG.*

A characterization of PCGs in terms of false twins is given by Calamoneri et al. [22].

Lemma 4.17. *A graph G with false twins u and v is a PCG if and only if the induced graph $G[V(G) \setminus \{u\}]$ is a PCG.*

Theorem 4.5 ([32, Theorem 2.8])**.** *Let n and m be positive integers. For an $m \times n$ matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ and a column vector $\mathbf{b} \in \mathbb{R}^m$, let $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$ be variables. Then exactly one of the following linear systems is feasible*

(i) $\mathbf{A}\mathbf{x} \leq \mathbf{b}$; *and*
(ii) $\mathbf{A}^{\mathrm{T}}\mathbf{y} \geq \mathbf{0},\ \mathbf{b}^{\mathrm{T}}\mathbf{y} < 0$.

4.3 LPs for Testing Plausibility of Configurations

Let $n \geq 1$ be an integer, and (G, T, σ, λ) be a configuration with $G \in \mathcal{G}_n$, $T \in \mathcal{T}_n$, $\sigma \in \Sigma(G, T)$ and $\lambda \in \Lambda_2(G, T, \sigma)$. Recall that (G, T, σ, λ) is plausible if and only if there are a weight function $w : E(T) \to \mathbb{R}_+$ and reals $d_{\min}, d_{\max} \in \mathbb{R}_+$ such that $G = \mathrm{PCG}(T, w, d_{\min}, d_{\max})$ and $d_{T,w}(\sigma(u), \sigma(v)) <$

$d_{\min}$ (resp., $d_{T,w}(\sigma(u),\sigma(v)) > d_{\max}$) holds for each pair $uv \in \overline{E}(G)$ with $\lambda(uv) = 0$ (resp., $\lambda(uv) = 1$), where with some adequate scaling on reals we can assume that $\min\{d_{\min} - d_{T,w}(\sigma(u),\sigma(v)) \mid uv \in \overline{E}(G), \lambda(uv) = 0\} \geq 1$ and $\min\{d_{T,w}(\sigma(u),\sigma(v)) - d_{\max} \mid uv \in \overline{E}(G), \lambda(uv) = 1\} \geq 1$. Then the condition on plausibility can be expressed as a linear system by regarding $w(e)$, $e \in E(T)$, $d_{\min}$ and $d_{\max}$ as non-negative variables. Observe that the next LP, $\mathrm{LP}(G,T,\sigma,\lambda)$ without an objective function is feasible if and only if (G,T,σ,λ) is plausible.

(a) Linear programming formulation $\mathrm{LP}(G,T,\sigma,\lambda)$
variables

$$\begin{aligned} & w(e) \geq 0, && \forall e \in E(T), \\ & d_{\min}, d_{\max} \geq 0, \end{aligned}$$

subject to

$$\begin{aligned} d_{\min} - d_{\max} &\leq 0, && && (4.3.1) \\ \sum_{e \in E_T(\sigma(u),\sigma(v))} w(e) - d_{\max} &\leq 0, && \forall uv \in E(G), && (4.3.2) \\ -\sum_{e \in E_T(\sigma(u),\sigma(v))} w(e) + d_{\min} &\leq 0, && \forall uv \in E(G), && (4.3.3) \\ \sum_{e \in E_T(\sigma(u),\sigma(v))} w(e) - d_{\min} &\leq -1, && \forall uv \in \overline{E}(G), \lambda(uv) = 0, && (4.3.4) \\ -\sum_{e \in E_T(\sigma(u),\sigma(v))} w(e) + d_{\max} &\leq -1, && \forall uv \in \overline{E}(G), \lambda(uv) = 1. && (4.3.5) \end{aligned}$$

When the $\mathrm{LP}(G,T,\sigma,\lambda)$ is feasible, any feasible solution to the LP will be an evidence based on which we can easily see that the graph G is a PCG by computing $\mathrm{PCG}(T, w, d_{\min}, d_{\max})$. However for an NPCG G, no configuration (G,T,σ,λ) is plausible and the $\mathrm{LP}(G,T,\sigma,\lambda)$ for any configuration with G is infeasible. To get an evidence to an NPCG G, we introduce another LP formulation $\mathrm{DLP}(G,T,\sigma,\lambda)$ so that it is feasible if and only if (G,T,σ,λ) is implausible. Then we can construct an evidence to an NPCG G by collecting a feasible solution to the feasible $\mathrm{DLP}(G,T,\sigma,\lambda)$ over all configurations (G,T,σ,λ), $T \in \mathcal{T}_n$, $\sigma \in \Sigma(G,T)$ and $\lambda \in \Lambda_2(G,T,\sigma)$. The $\mathrm{DLP}(G,T,\sigma,\lambda)$ is formulated based on the dual of the $\mathrm{LP}(G,T,\sigma,\lambda)$ as follows. The feasibility of this LP is discussed in Theorem 4.6.

(b) Linear programming formulation $\mathrm{DLP}(G,T,\sigma,\lambda)$

variables

$$
\begin{aligned}
& t \geq 0, && \\
& y_{uv}, z_{uv} \geq 0, && \forall uv \in E(G), \\
& y^{-}_{uv} \geq 0, && \forall uv \in \overline{E}(G), \lambda(uv) = 0, \\
& y^{+}_{uv} \geq 0, && \forall uv \in \overline{E}(G), \lambda(uv) = 1,
\end{aligned}
$$

subject to

$$
\begin{aligned}
& \sum_{\substack{uv \in E(G): \\ e \in E_T(\sigma(u),\sigma(v))}} (y_{uv} - z_{uv}) + \sum_{\substack{uv \in \overline{E}(G): \\ e \in E_T(\sigma(u),\sigma(v)), \lambda(uv)=0}} y^{-}_{uv} - \\
& \qquad \sum_{\substack{uv \in \overline{E}(G): \\ \forall e \in E_T(\sigma(u),\sigma(v)), \lambda(uv)=1}} y^{+}_{uv} \geq 0, \qquad e \in E(T), && (4.3.6) \\
& t + \sum_{uv \in E(G)} z_{uv} - \sum_{uv \in \overline{E}(G): \lambda(uv)=0} y^{-}_{uv} \geq 0, && (4.3.7) \\
& -t - \sum_{uv \in E(G)} y_{uv} + \sum_{uv \in \overline{E}(G): \lambda(uv)=1} y^{+}_{uv} \geq 0, && (4.3.8) \\
& - \sum_{uv \in \overline{E}(G): \lambda(uv)=0} y^{-}_{uv} - \sum_{uv \in \overline{E}(G): \lambda(uv)=1} y^{+}_{uv} \leq -1. && (4.3.9)
\end{aligned}
$$

Note that testing if $x < a$ holds precisely in a numerical computation with any precision tolerance is not possible. It is therefore the LHSs of (4.3.4), (4.3.5) and (4.3.9) are set to be at most -1 which should originally be strictly less than zero.

The LP(G, T, σ, λ) has $|E(T)| + 2$ variables and $\mathcal{O}(|L(T)|^2)$ constraints, while the DLP(G, T, σ, λ) has $\mathcal{O}(|L(T)|^2)$ variables and $|E(T)| + 3$ constraints.

Theorem 4.6. *For a given configuration* (G, T, σ, λ)*, exactly one of the linear programs* LP(G, T, σ, λ) *and* DLP(G, T, σ, λ) *is feasible.*

Proof. We apply Theorem 4.5 as follows. Observe that the linear system LP(G, T, σ, λ) with Eqs. (4.3.1) - (4.3.5) can be expressed as the following form with a $\{0, \pm 1\}$-coefficient matrix $\mathbf{A}$, a $\{0, -1\}$-vector $\mathbf{b}$ of coefficients and a vector $\mathbf{x}$ of non-negative variables:

$$\mathbf{A}\mathbf{x} \leq \mathbf{b}. \qquad (4.3.10)$$

By Theorem 4.5, it follows that either the linear system in Eq. (4.3.10) is feasible or the next linear system is feasible

$$
\begin{aligned}
& \mathbf{A}^{\mathrm{T}}\mathbf{y} \geq \mathbf{0}, && (4.3.11) \\
& \mathbf{b}^{\mathrm{T}}\mathbf{y} < 0. && (4.3.12)
\end{aligned}
$$

It can be easily verified that Eqs. (4.3.6) - (4.3.8) in DLP(G, T, σ, λ) are expressed as Eq. (4.3.11) and Eq. (4.3.9) in DLP(G, T, σ, λ) is expressed as $\mathbf{b}^{\mathrm{T}}\mathbf{y} \leq -1$. To prove the theorem, it suffices to show that the feasibility of the linear system with Eqs. (4.3.11) and (4.3.12) is equivalent to that of DLP(G, T, σ, λ). We can modify $\mathbf{b}^{\mathrm{T}}\mathbf{y} < 0$ into $\mathbf{b}^{\mathrm{T}}\mathbf{y} \leq -1$ without changing the feasibility. This is because for a feasible solution $\mathbf{y}$ to Eqs. (4.3.11) and (4.3.12) with $\mathbf{b}^{\mathrm{T}}\mathbf{y} = \varepsilon \in (-1, 0)$, the new vector $\mathbf{y}^* = \mathbf{y}/(-\varepsilon)$ is also feasible to Eqs. (4.3.11) and $\mathbf{b}^{\mathrm{T}}\mathbf{y} \leq -1$. □

The LP formulation LP(G, T, σ, λ) (resp., DLP(G, T, σ, λ)) outputs a proof for the plausibility (resp., implausibility) of the configuration (G, T, σ, λ) if it is feasible.

4.3.1 Comparison of the Proposed LPs

Calamoneri et al. [23] also introduced an LP formulation that confirms if a given graph is a PCG or not. The formulations LP(G, T, σ, λ) and DLP(G, T, σ, λ) differ from the LP formulation due to Calamoneri et al. [23] in the following points:

(1) The input of the LP by Calamoneri et al. [23] is a graph and a tree, while the input of our LPs is a graph, a tree, a bijection and a 2-coloring.

(2) The feasibility of the LP formulation introduced by Calamoneri et al. [23] implies that the given graph is a PCG. On the other hand, the feasibility of the LP(G, T, σ, λ) implies that the configuration (G, T, σ, λ) is plausible, and hence the graph G is a PCG. However, the infeasibility of the LP introduced by Calamoneri et al. [23] implies that the given graph is not a PCG. The infeasibility of LP(G, T, σ, λ) implies that the configuration (G, T, σ, λ) is implausible, and therefore the graph G is not a PCG due to T with bijection σ and 2-coloring λ. Additionally, DLP(G, T, σ, λ) is formulated such that it is feasible if and only if LP(G, T, σ, λ) is infeasible based on Gale's Theorem 4.5.

(3) If a graph G is a PCG, then the LP formulation by Calamoneri et al. [23] is feasible and outputs a weight function w, two reals $d_{\min}$ and $d_{\max}$ and a bijection $\sigma \in \Sigma(G, T)$ such that $G = \mathrm{PCG}(T, w, d_{\min}, d_{\max})$. If LP$(G, T, \sigma, \lambda)$ is feasible then a solution w, $d_{\min}$, $d_{\max}$ to LP(G, T, σ, λ) is such that the configuration (G, T, σ, λ) is plausible.

(4) For a fixed graph G with n vertices and tree T with n leaves, the LP introduced by Calamoneri et al. [23] has $\mathcal{O}(n^2) + |E(T)| + 2$ variables and $\mathcal{O}(n^4)$ constraints. On the other hand, the LP(G, T, σ, λ) (resp. DLP(G, T, σ, λ)) has $|E(T)| + 2$ (resp. $\mathcal{O}(n^2)$) variables and $\mathcal{O}(n^2)$ (resp. $|E(T)| + 3$) constraints.

4.4 A Sufficient PCG Condition

Let $G \in \mathcal{G}_n$ and $T \in \mathcal{T}_n$. This section formulates an ILP, $\mathrm{ILP}_{\mathrm{suff}}$ that can be used as a sufficient condition for G to be a PCG with witness tree T. In other words, the task is to find a bijection $\sigma : V(G) \to L(T)$, a weight function $w : E(T) \to \mathbb{R}_+$ and reals $d_{\min}$ and $d_{\max}$ such that σ is an isomorphism between G and $\mathrm{PCG}(T, w, d_{\min}, d_{\max})$. However, to formulate this as an ILP, we fix an integer $\alpha \geq 1$ and restrict the search domains for variables $w(e)$, $e \in E(T)$, $d_{\min}$ and $d_{\max}$ to non-negative reals bounded from above by α. As in the $\mathrm{LP}(G, T, \sigma, \lambda)$, we first prepare variables for a weight function w and reals $d_{\min}$ and $d_{\max}$, where $w(e) \in [0, \alpha]$, $e \in E(T)$ and two numbers $d_{\min}, d_{\max} \in [1, \alpha]$. Also we need to introduce variables for a bijection $\sigma \in \Sigma(G, T)$ and a 2-coloring $\lambda \in \Lambda_2(G, T, \sigma)$ and we use binary variables $\lambda(e) \in \{0, 1\}$, $e \in \overline{E}(G)$ to express a 2-coloring $\lambda : \overline{E}(G) \to \{0, 1\}$. Since G may not be a PCG with witness tree T, our ILP formulation must be able to express this case too. For this, we introduce a dummy leaf q to T, consider an extended mapping $\sigma : V(G) \to L(T) \cup \{q\}$, and prepare binary variables $\sigma_{us} \in \{0, 1\}$, $u \in V(G)$ and $s \in L(T) \cup \{q\}$. We formulate an ILP so that σ gives a bijection from $V(G)$ to $L(T)$ if and only if G is a PCG with witness tree T as follows:

(i) σ maps no two vertices to the same leaves in $L(T)$, whereas two or more vertices may be mapped to the dummy leaf q;

(ii) For each edge uv in G such that $\{\sigma(u), \sigma(v)\} \subseteq L(T)$, it holds that $d_{T,w}(\sigma(u), \sigma(v)) \in [d_{\min}, d_{\max}]$;

(iii) For each pair $uv \in \overline{E}(G)$ such that $\{\sigma(u), \sigma(v)\} \subseteq L(T)$, it holds that $d_{T,w}(\sigma(u), \sigma(v)) < d_{\min}$ (resp., $d_{T,w}(\sigma(u), \sigma(v)) > d_{\max}$) if $\lambda(uv) = 0$ (resp., $\lambda(uv) = 1$); and

(iv) Introduce an objective function that minimizes the number of vertices in $V(G)$ that are mapped by σ to the dummy leaf q. The optimal value becomes zero if and only if G is a PCG with witness tree T.

$\mathrm{ILP}_{\mathrm{suff}}(G, T, \alpha)$ is formulated below.

Integer linear programming formulation $\mathrm{ILP}_{\mathrm{suff}}(G,T,\alpha)$
variables

$$\begin{aligned}
&w(e) \geq 0, && \forall e \in E(T),\\
&d_{\min}, d_{\max} \geq 0, &&\\
&\sigma_{us} \in \{0,1\}, && \forall u \in V(G),\ s \in L(T) \cup \{q\},\\
&\lambda(e) \in \{0,1\}, && \forall e \in \overline{E}(G),
\end{aligned}$$

$$\min \sum_{u \in V(G)} \sigma_{uq}, \tag{4.4.13}$$

subject to

$$1 \leq d_{\min} \leq d_{\max} \leq \alpha, \tag{4.4.14}$$

$$w(e) \leq \alpha, \quad \forall e \in E(T), \tag{4.4.15}$$

$$\sum_{e \in E_T(s,t)} w(e) \geq d_{\min} - (2-\sigma_{us}-\sigma_{vt})n\alpha, \quad \forall uv \in E(G), s,t \in L(T), \tag{4.4.16}$$

$$\sum_{e \in E_T(s,t)} w(e) \leq d_{\max} + (2-\sigma_{us}-\sigma_{vt})n\alpha, \quad \forall uv \in E(G), s,t \in L(T), \tag{4.4.17}$$

$$\sum_{e \in E_T(s,t)} w(e) \leq d_{\min}-1 + (2-\sigma_{us}-\sigma_{vt}+\lambda(uv))n\alpha, \quad \forall uv \in \overline{E}(G), s,t \in L(T), \tag{4.4.18}$$

$$\sum_{e \in E_T(s,t)} w(e) \geq d_{\max}+1 - (3-\sigma_{us}-\sigma_{vt}-\lambda(uv))n\alpha, \quad \forall uv \in \overline{E}(G), s,t \in L(T), \tag{4.4.19}$$

$$\sum_{s \in L(T) \cup \{q\}} \sigma_{vs} = 1, \forall v \in V(G), \tag{4.4.20}$$

$$\sum_{v \in V} \sigma_{vs} \leq 1, \forall s \in L(T). \tag{4.4.21}$$

To avoid a possible numerical error in solving ILPs, a margin of 1 is introduced in conditions (4.4.18) and (4.4.19).

For a graph G with n vertices and a binary tree T with n leaves, $\mathrm{ILP}_{\mathrm{suff}}(G,T,\alpha)$ has $\mathcal{O}(n^2)$ binary variables, $\mathcal{O}(n)$ continuous variables, and $\mathcal{O}(n^4)$ constraints.

Lemma 4.18. *For any two positive integers n and α, a graph $G \in \mathcal{G}_n$ and a tree $T \in \mathcal{T}_n$, the following statements hold*

(i) $\mathrm{ILP}_{\mathrm{suff}}(G, T, \alpha)$ *is feasible.*

(ii) *If the optimal value to* $\mathrm{ILP}_{\mathrm{suff}}(G, T, \alpha)$ *is zero, then* G *is a PCG with a witness tree* T.

Proof. (i) Let the variables of $\mathrm{ILP}_{\mathrm{suff}}(G, T, \alpha)$ take values such that $d_{\min} = d_{\max} = 1$ and $\sigma_{vq} = 1$ ($v \in V(G)$) and all the other variables are zero. We see that Eqs. (4.4.14) and (4.4.15) are satisfied since $\alpha \geq 1$. Since $\sigma_{vs} = 1$ if and only if $s = q$, Eqs. (4.4.20) and (4.4.21) are satisfied. In each of Eqs. (4.4.16) - (4.4.19), the left-hand side is zero, and $\sigma_{us} = \sigma_{vt} = \lambda(uv) = 0$ and $n, \alpha \geq 1$ mean that the last term of the right-hand side is at most -2 ($\geq -n\alpha$) for Eqs. (4.4.16) and (4.4.18) (resp., at least 2 ($\leq n\alpha$) for Eqs. (4.4.17) and (4.4.19)). Since $d_{\min} = d_{\max} = 1$, we see that Eqs. (4.4.16) to (4.4.19) are satisfied.

(ii) Suppose that the objective value to $\mathrm{ILP}_{\mathrm{suff}}(G, T, \alpha)$ is zero; i.e., $\sigma_{uq} = 0$ for all vertices $u \in V(G)$. Then by Eq. (4.4.20), each $u \in V(G)$ has exactly one leaf $s \in L(T)$ such that $\sigma_{uv} = 1$. Further, Eq. (4.4.21) implies that the mapping σ is an injection to the set $L(T)$. Thus, by using the fact that $|V(G)| = |L(T)|$ it holds that σ is a bijection between $V(G)$ and $L(T)$. Let $\{u, v\}$ be a pair of vertices in G, which has a unique pair of leaves $s, t \in L(T)$ such that $\sigma_{us} = \sigma_{vt} = 1$ due to the bijection σ. When $uv \in E(G)$, Eqs. (4.4.16) and (4.4.17) imply $d_{T,w}(s,t) \geq d_{\min}$ and $d_{T,w}(s,t) \leq d_{\max}$, respectively. When $uv \in \overline{E}(G)$ and $\lambda(uv) = 0$ (resp., $uv \in \overline{E}(G)$ and $\lambda(uv) = 1$), Eq. (4.4.18) (resp., Eq. (4.4.19)) indicates $d_{T,w}(s,t) \leq d_{\min} - 1 < d_{\min}$ (resp., $d_{T,w}(s,t) \geq d_{\max} + 1 > d_{\max}$). Therefore we see that G is isomorphic to $\mathrm{PCG}(T, w, d_{\min}, d_{\max})$, as required. □

It should be noted that, for a triplet (G, T, α) if the objective value to $\mathrm{ILP}_{\mathrm{suff}}(G, T, \alpha)$ is not equal to zero, then it does not necessarily imply that G is not a PCG. This is because of the restriction on the search domains of $d_{\min}, d_{\max} \in [1, \alpha]$ and $w(e) \in [0, \alpha]$ for each $e \in E(T)$, while G can be a PCG with T as its witness tree for other values $d_{\min}, d_{\max}$ and w from the original domain $[0, \infty)$.

4.5 Characterization of 2-colorings of MIC4s

By the known fact that a graph is a PCG if and only if each of its induced subgraphs is a PCG [10], it follows that a configuration is plausible if and only if all of its subconfigurations are plausible. Thus, the absence of MIC4 subconfigurations is a necessary condition for a configuration to be plausible. Therefore, we characterize MIC4 configurations so that we can efficiently conclude if a configuration is MIC4-free or not in our method to enumerate PCGs in Chapter 5.

We computed all non-isomorphic MICs with $n = 4$, and we have the following result.

Lemma 4.19. *There are exactly 59 non-isomorphic MICs with four vertices.*

Let $T \in \mathcal{T}_4$. There are two vertices namely s and t in $V(T)$ which are of degree 3. Let a and b (resp., c and d) denote the children of the vertex s (resp., t). With these notations, T is illustrated in Fig. 4.3, and Fig. 4.4 illustrates a maximal set of 59 MICs. We shows that all MICs with $n = 4$ can be characterized

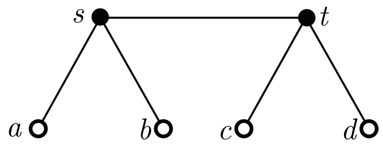


Figure 4.3. The binary tree T with $V(T) = \{a, b, c, d, s, t\}$, $L(T) = \{a, b, c, d\}$ and $E(T) = \{as, bs, ct, dt, st\}$.

compactly with a set of three inequalities on 3-colorings ρ, where a pair of a graph G and a 2-coloring $\lambda : \overline{E}(G) \to \{0, 1\}$ is encoded into a 3-coloring $\rho : \binom{V(G)}{2} \to \{1, 2, 3\}$ so that

$$\rho(uv) := 2,\ uv \in E(G) \text{ and } \rho(uv) := 2\lambda(uv) + 1,\ uv \in \overline{E}(G).$$

Just by checking the inequalities, we will easily see if a given configuration is MIC4-free or not without referring to the list of 59 MICs. In fact, the new characterization can be obtained by inspecting the structure of each of the 59 MICs, but we here give a direct proof to the characterization based on the definition of MICs.

For a pair $\delta = (\delta_1, \delta_2)$ of reals $0 \le \delta_1 \le \delta_2$, we define a function $f_\delta : \mathbb{R} \to \{1, 2, 3\}$ so that

$$f_\delta(x) \triangleq \begin{cases} 1 & \text{if } x < \delta_1, \\ 2 & \text{if } \delta_1 \le x \le \delta_2, \\ 3 & \text{if } \delta_2 < x. \end{cases}$$

Let $T \in \mathcal{T}_4$ denote a binary tree with four leaves, where we denote $V(T) = \{a, b, c, d, s, t\}$, $L(T) = \{a, b, c, d\}$ and $E(T) = \{as, bs, ct, dt, st\}$. Let $\rho : \binom{L(T)}{2} \to \{1, 2, 3\}$ be a 3-coloring, where we denote $\rho(\{u, v\})$ for $u, v \in \{a, b, c, d\}$ simply by ρ_{uv}. We call a pair (w, δ) of a weight function $w : E(T) \to \mathbb{R}_+$ and a pair $\delta = (\delta_1, \delta_2)$ of reals $0 \le \delta_1 \le \delta_2$ *admissible with respect to* ρ if for each $\{u, v\} \in \binom{L(T)}{2}$ it holds that $f_\delta(d_{T,w}(u, v)) = \rho_{uv}$. For any admissible pair (w, δ) with respect to ρ, $G = \text{PCG}(T, w, \delta_1, \delta_2)$ is a PCG such that $\rho(uv) = 2$ for each pair $uv \in E(G)$ and $\rho(uv) = 1$ (resp., $\rho(uv) = 3$) for each pair $uv \in \overline{E}(G)$ with $d_{T,w}(u, v) < \delta_1$ (resp., $d_{T,w}(u, v) > \delta_2$). On the other hand, evey PCG $G = \text{PCG}(T, w, d_{\min}, d_{\max})$ has

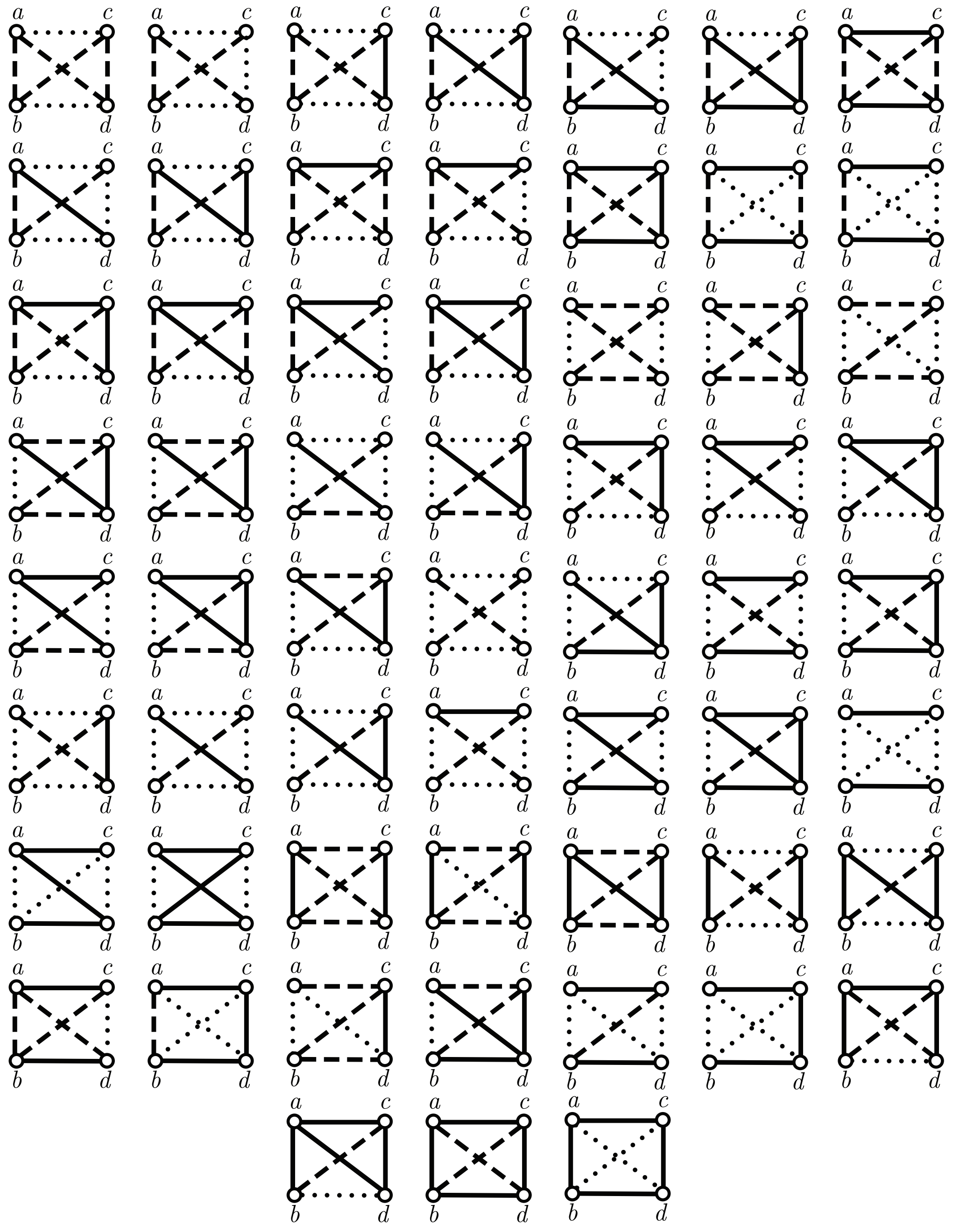


Figure 4.4. A maximal set of 59 non-isomorphic MICs with four vertices. Each graph represents an MIC (G, T, I, λ) with four vertices, where we consider $V(G) = L(T)$, the heavy lines correspond to the edges in the graph G, while the heavy dashed and doted lines correspond to the pair of non-adjacent vertices uv in the graph G such that $\lambda(uv) = 0$ and $\lambda(uv) = 1$, respectively.

a pair $(w, \delta = (d_{\min}, d_{\max}))$ admissible with respect to a 3-coloring ρ such that $\rho(uv) = 2$, $uv \in E(G)$ and $\rho(uv) = 1$ (resp., $\rho(uv) = 3$) for each pair $uv \in \overline{E}(G)$ with $d_{T,w}(u, v) < d_{\min}$ (resp., $d_{T,w}(u, v) > d_{\max}$).

Lemma 4.20. *Let $T \in \mathcal{T}_4$ with $V(T) = \{a, b, c, d, s, t\}$, $L(T) = \{a, b, c, d\}$ and $E(T) = \{as, bs, ct, dt, st\}$, and $\rho : \binom{L(T)}{2} \to \{1, 2, 3\}$. Then there are a weight function $w : E(T) \to \mathbb{R}_+$ and a pair $\delta = (\delta_1, \delta_2)$ of reals $0 < \delta_1 < \delta_2$ such that $\{d_{T,w}(u, v) \mid \{u, v\} \in \binom{L(T)}{2}\} \cap \{0, \delta_1, \delta_2\} = \emptyset$ and (w, δ) is an admissible pair with respect to ρ if and only if ρ does not satisfy any of Eqs.* (4.5.22) - (4.5.24):

$$|\rho_{ab} - \rho_{cd}|, |\rho_{ad} - \rho_{bc}| \le 1,\ \rho_{ab} + \rho_{cd} - 2 \ge \rho_{ad} + \rho_{bc}; \tag{4.5.22}$$

$$|\rho_{ab} - \rho_{cd}|, |\rho_{ac} - \rho_{bd}| \le 1,\ \rho_{ab} + \rho_{cd} - 2 \ge \rho_{ac} + \rho_{bd}; \tag{4.5.23}$$

$$|\rho_{ac} - \rho_{bd}|, |\rho_{ad} - \rho_{bc}| \le 1,\ |\rho_{ac} + \rho_{bd} - (\rho_{ad} + \rho_{bc})| \ge 2. \tag{4.5.24}$$

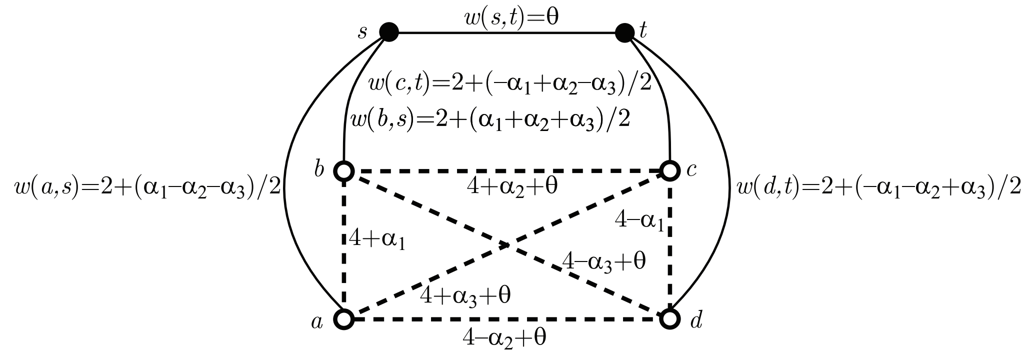
Proof. The if-part: Let ρ satisfy one of Eqs. (4.5.22) - (4.5.24). To derive a contradiction, we assume that there is a pair (w, δ) that satisfies the condition in the lemma. Define $\delta_0 := 0$ and $\delta_3 := \delta_2 + 3 \max\{w(e) \mid e \in E(T)\}$. Note that $\delta_{\rho_{ab}-1} < d_{T,w}(a, b) < \delta_{\rho_{ab}}$ holds for ρ_{ab} and $\delta_{\rho_{ad}-1} < d_{T,w}(a, d) < \delta_{\rho_{ad}}$ holds for ρ_{ad}. Analogously with ρ_{cd}, ρ_{bc}, ρ_{ac} and ρ_{bd}.

First consider the case when ρ satisfies Eq. (4.5.22). Note that $\delta_{\rho_{ab}-1} + \delta_{\rho_{cd}-1} < d_{T,w}(a, b) + d_{T,w}(c, d) < \delta_{\rho_{ad}} + \delta_{\rho_{bc}}$. From this, we see that $\min\{\rho_{ab}, \rho_{cd}\} \le \max\{\rho_{ad}, \rho_{bc}\}$ and $\{\rho_{ab} - 1, \rho_{cd} - 1\} \ne \{\rho_{ad}, \rho_{bc}\}$. This and $|\rho_{ab} - \rho_{cd}|, |\rho_{ad} - \rho_{bc}| \le 1$ mean that $\rho_{ab} + \rho_{cd} - 1 \le \rho_{ad} + \rho_{bc}$. This contradicts $\rho_{ab} + \rho_{cd} - 2 \ge \rho_{ad} + \rho_{bc}$.

Next consider the case when ρ satisfies Eq. (4.5.23). This case can be treated analogously with the above case by exchanging the role of vertices c and d.

Finally, consider the case when ρ satisfies Eq. (4.5.24). It is easy to observe that $\delta_{\rho_{ad}-1} + \delta_{\rho_{bc}-1} < d_{T,w}(a, c) + d_{T,w}(b, d) < \delta_{\rho_{ac}} + \delta_{\rho_{bd}}$. This implies that $\min\{\rho_{ad}, \rho_{bc}\} \le \max\{\rho_{ac}, \rho_{bd}\}$ and $\{\rho_{ad} - 1, \rho_{bc} - 1\} \ne \{\rho_{ac}, \rho_{bd}\}$, where the vertices c and d can exchange their role. Thus, it follows from this fact and $|\rho_{ac} - \rho_{bd}|, |\rho_{ad} - \rho_{bc}| \le 1$ that $\rho_{ad} + \rho_{bc} - 1 \le \rho_{ac} + \rho_{bd}$, and $\rho_{ac} + \rho_{bd} - 1 \le \rho_{ad} + \rho_{bc}$. This contradicts $|\rho_{ac} + \rho_{bd} - (\rho_{ad} + \rho_{bc})| \ge 2$, which completes the if-part.

The only if-part: We prove that, for any 3-coloring $\rho : \binom{L(T)}{2} \to \{1, 2, 3\}$ that satisfies none of Eqs. (4.5.22) - (4.5.24), an admissible pair (w, δ) can be chosen. For this, we will distinguish five different types (C1) - (C5) among all such 3-colorings ρ, define the weight w of T with four parameters $\alpha_1, \alpha_2, \alpha_3$ and θ, and show that $(w, \delta = (\delta_1, \delta_2))$ becomes admissible with respect to each type of 3-coloring ρ by choosing adequate values to $\alpha_1, \alpha_2, \alpha_3$, θ, δ_1 and δ_2.



$w(s,t)=\theta$
s
t
$w(c,t)=2+(-\alpha_1+\alpha_2-\alpha_3)/2$
$w(b,s)=2+(\alpha_1+\alpha_2+\alpha_3)/2$
$w(a,s)=2+(\alpha_1-\alpha_2-\alpha_3)/2$
$w(d,t)=2+(-\alpha_1-\alpha_2+\alpha_3)/2$
b
c
$4+\alpha_2+\theta$
$4-\alpha_1$
$4+\alpha_1$
$4-\alpha_3+\theta$
$4+\alpha_3+\theta$
a
d
$4-\alpha_2+\theta$

Figure 4.5. Tree T with $V(T) = \{a, b, c, d, s, t\}$, $L(T) = \{a, b, c, d\}$ and $E(T) = \{as, bs, ct, dt, st\}$, and edge weight function w. For each pair of leaves $\{i, j\} \subseteq L(T)$, the distance $d_{T,w}(i, j)$ between i and j is given along the dashed line between i and j.

Let $\alpha_1, \alpha_2, \alpha_3$ be distinct non-zero reals such that $-2/3 < \alpha_i < 2/3$, $i = 1, 2, 3$, and θ be a non-negative real. With these reals, we set a weight function $w : E(T) \to \mathbb{R}_+$ to be

$$\begin{aligned}
&w(a, s) = 2 + (\alpha_1 - \alpha_2 - \alpha_3)/2, \quad w(b, s) = 2 + (\alpha_1 + \alpha_2 + \alpha_3)/2,\\
&w(c, t) = 2 + (-\alpha_1 + \alpha_2 - \alpha_3)/2, \quad w(d, t) = 2 + (-\alpha_1 - \alpha_2 + \alpha_3)/2,\\
&w(s, t) = \theta,
\end{aligned}$$

from which we see that

$$\begin{aligned}
&d_{T,w}(a, b) = 4 + \alpha_1, && d_{T,w}(c, d) = 4 - \alpha_1, && d_{T,w}(a, d) = 4 - \alpha_2 + \theta,\\
&d_{T,w}(b, c) = 4 + \alpha_2 + \theta, && d_{T,w}(a, c) = 4 + \alpha_3 + \theta, && d_{T,w}(b, d) = 4 - \alpha_3 + \theta.
\end{aligned}$$

(see Fig. 4.5). Denote

$$\begin{aligned}
&d_1 = \min\{d_{T,w}(a, b), d_{T,w}(c, d)\}, \quad d'_1 = \max\{d_{T,w}(a, b), d_{T,w}(c, d)\},\\
&d_2 = \min\{d_{T,w}(a, d), d_{T,w}(b, c)\}, \quad d'_2 = \max\{d_{T,w}(a, d), d_{T,w}(b, c)\},\\
&d_3 = \min\{d_{T,w}(a, c), d_{T,w}(b, d)\}, \quad d'_3 = \max\{d_{T,w}(a, c), d_{T,w}(b, d)\},
\end{aligned}$$

where we have $\{d_1, d'_1\} = \{4 + \alpha_1, 4 - \alpha_1\}$, $\{d_i, d'_i\} = \{4 + \alpha_i + \theta, 4 - \alpha_i + \theta\}$, $i = 2, 3$. For a pair $\delta = (\delta_1, \delta_2)$, denote

$$\rho_i = f_\delta(d_i), \ \rho'_i = f_\delta(d'_i), \ i = 1, 2, 3. \tag{4.5.25}$$

The assumption that a given 3-coloring ρ does not satisfy any of Eqs. (4.5.22) - (4.5.24) can be stated as the following conditions in terms of $\rho_i, \rho'_i \in \{1, 2, 3\}$,

$i = 1, 2, 3$:

$$\text{if } (\rho_1, \rho'_1), (\rho_2, \rho'_2) \neq (1, 3) \text{ then } \rho_1 + \rho'_1 - 1 \leq \rho_2 + \rho'_2; \tag{4.5.26}$$

$$\text{if } (\rho_1, \rho'_1), (\rho_3, \rho'_3) \neq (1, 3) \text{ then } \rho_1 + \rho'_1 - 1 \leq \rho_3 + \rho'_3; \tag{4.5.27}$$

$$\text{if } (\rho_2, \rho'_2), (\rho_3, \rho'_3) \neq (1, 3) \text{ then } |\rho_2 + \rho'_2 - (\rho_3 + \rho'_3)| \leq 1. \tag{4.5.28}$$

Let $\rho = \{\rho_i, \rho'_i \in \{1, 2, 3\} \mid i = 1, 2, 3\}$ be a 3-coloring that satisfies Eqs. (4.5.26) - (4.5.28). In what follows, we show that for an adequate choice of $(\alpha_1, \alpha_2, \alpha_3, \theta, \delta_1, \delta_2)$, the pair (w, δ) becomes admissible (i.e., (w, δ) actually satisfies Eq. (4.5.25)). To facilitate this, we classify the 3-colorings $\rho = \{\rho_i, \rho'_i \in \{1, 2, 3\} \mid i = 1, 2, 3\}$ that satisfy Eqs. (4.5.26) - (4.5.28) into the following five types (C1) - (C5) of disjoint sets of 3-colorings.

(C1) $(\rho_i, \rho'_i) = (\rho_j, \rho'_j) = (1, 3)$ for some two indices $i, j \in \{1, 2, 3\}$, and $(\rho_k, \rho'_k) \in \{(p, q) \mid 1 \leq p \leq q \leq 3\}$, for the index $k \in \{1, 2, 3\} \setminus \{i, j\}$. There are six such possible 3-colorings;

(C2) $(\rho_i, \rho'_i) = (1, 3)$ for exactly one index $i \in \{1, 2, 3\}$ and $(\rho_j, \rho'_j), (\rho_k, \rho'_k) \in \{(p, q) \mid 1 \leq p \leq q \leq 3\} \setminus \{(1, 3)\}$ and $|\rho_j + \rho'_j - (\rho_k + \rho'_k)| \leq 1$ for the indices $\{j, k\} = \{1, 2, 3\} \setminus \{i\}$. There are nine 3-colorings of this type;

(C3) $(\rho_1, \rho'_1), (\rho_2, \rho'_2), (\rho_3, \rho'_3) \in \{(p, q) \mid 1 \leq p \leq q \leq 3\} \setminus \{(1, 3)\}$ and $|\rho_i + \rho'_i - (\rho_j + \rho'_j)| \leq 1$ for each pair $\{i, j\} \subseteq \{1, 2, 3\}$. There are 13 such 3-colorings;

(C4) $(\rho_i, \rho'_i) = (1, 3)$ holds for exactly one index $i \in \{1, 2, 3\}$, and the pairs (ρ_j, ρ'_j) and (ρ_k, ρ'_k) for the indices $\{j, k\} = \{1, 2, 3\} \setminus \{i\}$ do not satisfy the condition of (C2); i.e., it holds that $|\rho_j + \rho'_j - (\rho_k + \rho'_k)| \geq 2$. Let $(\rho_2, \rho'_2) = (1, 3)$, where by Eq. (4.5.27) we only need to consider the case $\rho_1 + \rho'_1 + 2 \leq \rho_3 + \rho'_3$. There are six possible such 3-colorings; and

(C5) $(\rho_i, \rho'_i) \neq (1, 3)$ for all indices $i \in \{1, 2, 3\}$, and the pairs $(\rho_1, \rho'_1), (\rho_2, \rho'_2), (\rho_3, \rho'_3)$ do not satisfy the condition of (C3); i.e., there exists a pair of indices $\{i, j\} \subseteq \{1, 2, 3\}$ such that $|\rho_i + \rho'_i - (\rho_j + \rho'_j)| \geq 2$ holds. Assume that $|\rho_1 + \rho'_1 - (\rho_i + \rho'_i)| \geq 2$ for some $i \in \{2, 3\}$, implying $\rho_1 + \rho'_1 + 2 \leq \rho_i + \rho'_i$, and hence $\rho_1 + \rho'_1 + 1 \leq \rho_j + \rho'_j$ for $j \in \{2, 3\} \setminus \{i\}$ by $|\rho_2 + \rho'_2 - (\rho_3 + \rho'_3)| \leq 1$. There are 13 such 3-colorings.

For a 3-coloring $\rho = \{\rho_i, \rho'_i \in \{1, 2, 3\} \mid i = 1, 2, 3\}$ of types (C1) - (C3), we first set $\theta = 0$, which implies that $\{d_1, d_2, d_3\} = \{4 + \min\{\alpha_1, -\alpha_1\}, 4 + \min\{\alpha_2, -\alpha_2\}, 4 + \min\{\alpha_3, -\alpha_3\}\}$ by the definition of w and $d_i, d'_i, i = 1, 2, 3$. This means that for any choice of three distinct indices $i, j, k \in \{1, 2, 3\}$, we can attain a total ordering $d_i < d_j < d_k < d'_k < d'_j < d'_i$, as illustrated in Fig. 4.6(a), by adjusting the values of α_1, α_2 and α_3.

For type (C1), we choose $\delta = (\delta_1, \delta_2)$ with $d_j < \delta_1 < \delta_2 < d'_j$ so that it holds $(\rho_i = f_\delta(d_i), \rho'_i = f_\delta(d'_i)) = (\rho_j = f_\delta(d_j), \rho'_i = f_\delta(d'_j)) = (1, 3)$. Now we see that six different choices of $\delta = (\delta_1, \delta_2)$ among all such δ can attain all possible values (p, q) with $1 \leq p \leq q \leq 3$ for $(\rho_k = f_\delta(d_k), \rho'_k = f_\delta(d'_k))$, as illustrated in Fig. 4.6.

For type (C2), we choose $\delta = (\delta_1, \delta_2)$ with $\delta_1 < \delta_2$ such that $d_i < \delta_1 < d_j$ or $d'_j < \delta_2 < d'_i$ to attain $(\rho_i = f_\delta(d_i), \rho'_i = f_\delta(d'_i)) = (1, 3)$. Then, we see that nine different choices of $\delta = (\delta_1, \delta_2)$ can attain all possible values (p, q) with $1 \leq p \leq q \leq 3$ for $(\rho_k = f_\delta(d_k), \rho'_k = f_\delta(d'_k))$ and $(\rho_j = f_\delta(d_j), \rho'_j = f_\delta(d'_j))$ of type (C2) (see Fig. 4.6).

For type (C3), we choose $\delta = (\delta_1, \delta_2)$ with $\delta_1 < \delta_2$ such that $\delta_1 < d_i$ or $\delta_2 > d'_i$ to attain $(\rho_i = f_\delta(d_i), \rho'_i = f_\delta(d'_i))$, $(\rho_k = f_\delta(d_k), \rho'_k = f_\delta(d'_k))$ and $(\rho_j = f_\delta(d_j), \rho'_j = f_\delta(d'_j))$. Then we can verify that there are thirteen different choices of $\delta = (\delta_1, \delta_2)$ that attain all possible values (p, q) with $1 \leq p \leq q \leq 3$ for $(\rho_i = f_\delta(d_i), \rho'_i = f_\delta(d'_i))$, $(\rho_k = f_\delta(d_k), \rho'_k = f_\delta(d'_k))$ and $(\rho_j = f_\delta(d_j), \rho'_j = f_\delta(d'_j))$ of type (C3) (see Fig. 4.6).

Consider the type (C4). By choosing values $\alpha_1 = 0.1, \alpha_2 = 0.6, \alpha_3 = 0.2$, and $\theta = 0.4$, We can attain a total ordering $d_2 < d_1 < d'_1 < d_3 < d'_3 < d'_2$, as illustrated in Fig. 4.6(b). Then we choose $\delta = (\delta_1, \delta_2)$ with $d_2 < \delta_1 < \delta_2 < d'_2$ to attain $(\rho_1 = f_\delta(d_1), \rho'_1 = f_\delta(d'_1))$ and $(\rho_3 = f_\delta(d_3), \rho'_3 = f_\delta(d'_3))$. From Fig. 4.7(a), we see that six different choices of $\delta = (\delta_1, \delta_2)$ can attain all such (p, q) with $1 \leq p \leq q \leq 3$ for $(\rho_1 = f_\delta(d_1), \rho'_1 = f_\delta(d'_1))$ and $(\rho_3 = f_\delta(d_3), \rho'_3 = f_\delta(d'_3))$.

Finally consider the type (C5). By setting values $\alpha_1 = 0.1$, $\theta = 0.4$ and $\{\alpha_2, \alpha_3\} = \{0.2, 0.3\}$, we can attain a total ordering $d_1 < d'_1 < d_i < d_j < d'_j < d'_i$. We choose $\delta = (\delta_1, \delta_2)$ with $\delta_1 < \delta_2$ such that $d'_1 < \delta_1 < d_i$, $d_1 < \delta_1 < d'_1$ or $\delta_1 < d_1$ to attain $(\rho_i = f_\delta(d_i), \rho'_i = f_\delta(d'_i))$ and $(\rho_j = f_\delta(d_j), \rho'_j = f_\delta(d'_j))$. From Fig. 4.7(b), we see that thirteen different choices of $\delta = (\delta_1, \delta_2)$ can attain (p, q) with $1 \leq p \leq q \leq 3$ for $(\rho_i = f_\delta(d_i), \rho'_i = f_\delta(d'_i))$ and $(\rho_j = f_\delta(d_j), \rho'_j = f_\delta(d'_j))$.

Hence for each 3-coloring $\rho = \{\rho_i, \rho'_i \in \{1, 2, 3\} \mid i = 1, 2, 3\}$ that satisfies Eqs. (4.5.26) - (4.5.28) we have demonstrated that a weight function w and a pair of non-negative real values $\delta = (\delta_1, \delta_2)$ exist such that Eq. (4.5.25) is satisfied, which completes the proof.

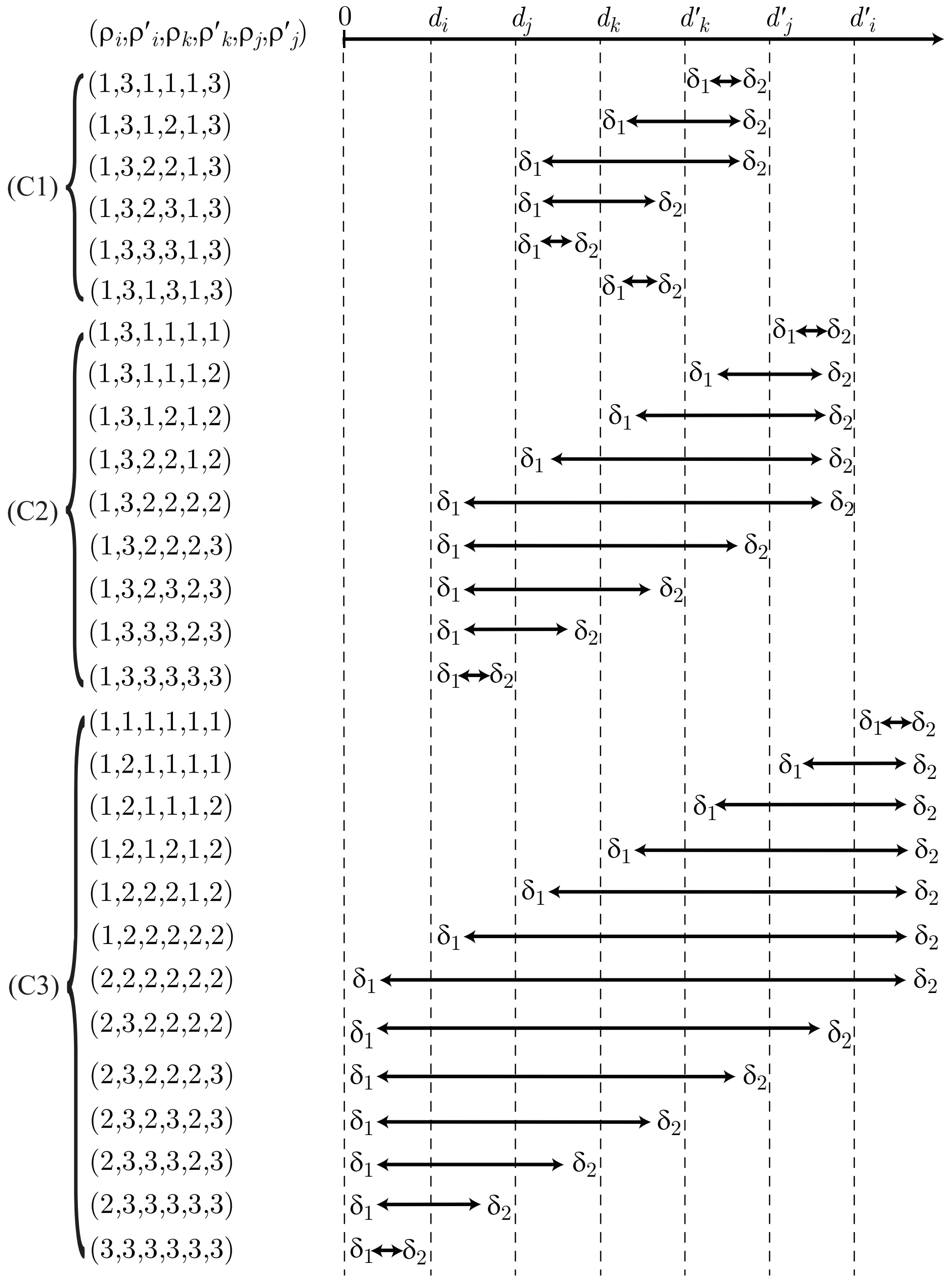


Figure 4.6. An illustration of possible choices for $\delta = (\delta_1, \delta_2)$ over the interval $[0, \infty)$ that can attain all possible values for ρ for each of the types (C1) to (C3). Each such $\delta = (\delta_1, \delta_2)$ is depicted as a line segment with left and right arrowheads joining points δ_1 and δ_2, with the respective value of ρ given as a 6-tuple on the left-hand side, subdivided by type (C1) to (C3). It is taken without loss of generality that $d_i < d'_i < d_j < d'_j < d_k < d'_k$.

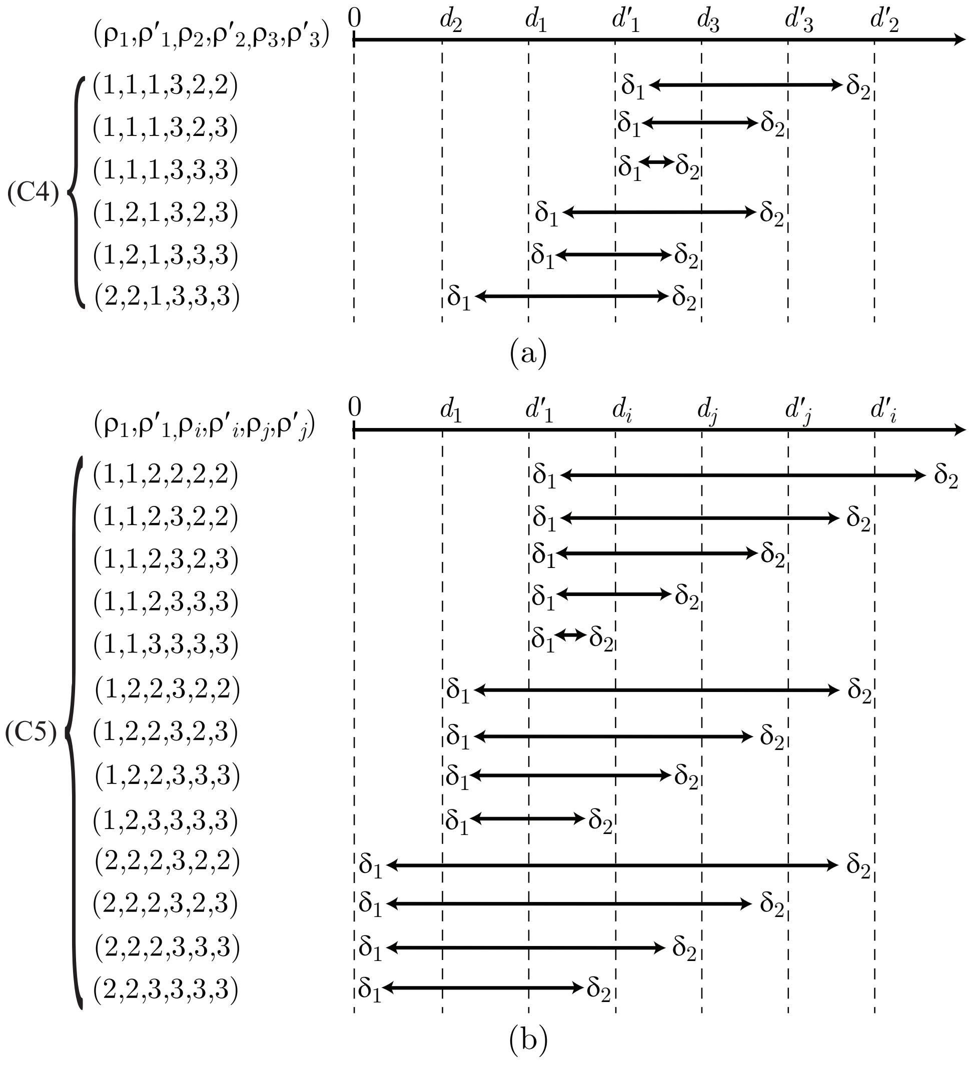


Figure 4.7. An illustration of possible choices for δ that can attain all possible values of ρ for each of types (C4) and (C5) not already included in types (C1)-(C3). Each such $\delta = (\delta_1, \delta_2)$ is depicted as a line segment with left and right arrowheads joining points δ_1 and δ_2. For each choice of δ the resulting assignment of ρ is given as a 6-tuple on the left-hand side. (a) Possible choices for δ for type (C4); (b) Possible choices for δ for type (C5).

□

We remark that Lemma 4.19 can be deduced from Lemma 4.20 by doing some case analysis.

4.6 Concluding Remarks

For a given configuration, we proposed two LP formulations exactly one of them is feasible to confirm if a given graph is a PCG or not due to the given configuration. We proposed a sufficient condition to efficiently test if a given graph is PCG by using an ILP formulation. The aim of this ILP formulation is to construct a plausible configuration and weight bounded above by a given positive integer. We computed all MICs with four vertices and found that there are exactly 59 MIC4s. Furthermore, we gave a characterization of MIC4 configurations in terms of a system of linear equations which can be used to efficiently test if a configuration is an MIC4-free.

We have proposed a sufficient condition expressed as the optimal value of the objective function of a certain ILP for a graph G to be a PCG with a witness tree T. If the sufficient condition is satisfied, this eliminates the need to further check all possible exponentially many configurations for the graph G and the tree T. However, a negative answer is inconclusive, and therefore it would be of interest to derive an efficiently verifiable necessary and sufficient condition that G is a PCG with a witness tree T.

Another interesting direction for future research is to characterize MICs on $k \geq 5$ vertices, and use them to test if a given configuration is an MICk-free configuration.

5 A Method to Enumerate Pairwise Compatibility Graphs

5.1 Introduction

A graph is called a pairwise compatibility graph (PCG) if there exists an edge-weighted tree and a one-to-one correspondence between the leaves of the tree and the vertices of the graph, such that there is an edge between two vertices in the graph if and only if the distance between their corresponding vertices in the tree is within a given interval. Recall that for a graph with $n \geq 1$ vertices, a configuration is defined to be a tuple that consists of the graph, a tree with n leaves, a correspondence between the vertices in the graph and the leaves in the tree, and a bi-partition of all pairs of non-adjacent vertices in the graph. A configuration is plausible if there exist a weight function and a closed interval such that the tree in the configuration is a witness of the graph, and one bi-partition class contains the pairs of non-adjacent vertices in the graph whose distance in the tree is to the left of the interval, and the other bi-partition class to the right. Observe that to show that a graph with $n \geq 1$ vertices is not a PCG it is necessary to exclude all configurations with the graph and real valued weight functions. This leads to the following three difficulties in enumerating all PCGs with a given number of vertices:

(i) Large number of configurations: Although the numbers of graphs, trees, correspondences, and bi-partitions are finite, the total number of configurations increases exponentially with the increase in n. For instance, when n is $7, 8$ and 9 there are approximately 2×10^{11}, 4×10^{14}, and 3×10^{18} configurations, respectively.

(ii) Infinite search space: There exists an infinite space of possible assignments of weights that needs to be excluded to confirm that a graph is not a PCG for a fixed configuration.

(iii) Finite size evidence: When a graph is a PCG then its witness tree is evidence to this. However, when a graph is not a PCG, then due to the infinite search space of edge weights it is a challenging task to construct finite size evidence that shows that a given graph is not a PCG.

These difficulties eventually necessitate a non-trivial method that can efficiently handle the problem of large number of configurations, infinite search space, and construction of finite size evidence to prove that a graph is not a PCG.

In this chapter, we propose a method to enumerate all PCGs with a given number of vertices. Our method consists of two main phases: (I) Graph screening; and (II) Constructing evidence based on linear programming (LP). The aim of (I) is to remove some graphs for which we do not need to further inspect configurations, to handle the difficulty (i), the large number of configurations. To achieve this, in phase (I) we use three methods: (I-1) Removing some graphs; (I-2) A PCG generator; and (I-2) Constructing plausible configurations. In (I-1), we eliminate some graphs G that can be checked to see if G is a PCG or not directly based on some observations. In (I-2), we heuristically generate all PCGs for a given weighted tree taking into account the tree symmetries to avoid repeatedly generating the same PCGs. In (I-3), we try to construct a plausible configuration with weights bounded from above, using a system of linear inequalities.

In phase (II), for each graph that is left after phase (I), we construct finite size evidence whether the graph is a PCG or not. Recall that a large number of configurations poses a difficulty, and therefore we only consider those configurations that satisfy a certain necessary condition to be plausible, i.e., MIC4-free configurations. In this phase, we propose two branch-and-bound algorithms to generate all MIC4-free configurations. Then to handle difficulties (ii) and (iii), we test each of the enumerated essential configurations by means of solving a linear program which is feasible if and only if the given configuration is plausible. As a definite proof to the infeasibility of the linear program, we use another linear program which is feasible if and only if the first one is not. Thus, we get finite size evidence as proof that the graph is a PCG or not. By using this method, we enumerated all PCGs with eight and nine vertices.

The rest of the chapter is organized as follows: We give a detailed explanation of our 2-phase PCG enumeration method in Section 5.2. In Section 5.2.1, we discuss phase (I) of our method and propose a PCG generator. In Section 5.2.2, we discuss phase (II) of our enumeration method where we propose a branch-and-bound algorithm to generate MIC4-free configurations. In Section 5.3, we propose another branch-and-bound algorithm to enumerate MIC4-free configurations that we use in phase (II). We discuss experimental results for enumerating PCGs with eight and nine vertices in Sections 5.2.3 and 5.3.1, respectively. Concluding remarks and future directions are given in Section 5.4.

5.2 A PCG Enumeration Method

We propose a PCG enumeration method to generate all PCGs with a given number of vertices. This is a 2-phase method, graphs screening phase and constructing evidence phase which are discussed in detail in Sections 5.2.1 and 5.2.2, respectively.

5.2.1 Phase I: Graph Screening

In this section, we describe phase (I) of our enumeration method, where in order to reduce our computational effort, we try to collect as many as possible graphs that are PCGs. This phase consists of three sub-phases: (I-1) Removal of graphs; (I-2) A PCG generator; and (I-3) Constructing plausible configurations which are explained below.

Phase I-1: Removal of Graphs

Assuming that all PCGs in $\mathcal{G}_t$ with $t < n$ are known, the first phase decreases the number of graphs $\mathcal{G}_n$ by eliminating graphs G that can be checked to see if G is a PCG or not directly based on some observations such as Lemma 4.16. Let $\mathcal{G}'_n := \mathcal{G}_n$. Then this phase executes the following three types of procedures:

(i) This step removes from $\mathcal{G}'_n$ all graphs $G \in \mathcal{G}_n$ such that G contains an MNPCG as a proper induced subgraph, where such a graph is an NPCG by Lemma 4.13.

(ii) This step removes from $\mathcal{G}'_n$ all non-biconnected graphs, since we easily see that any non-biconnected graph G is a PCG if and only if each of the biconnected components is a PCG by Lemma 4.16.

(iii) In this step, we remove graphs with false twins from $\mathcal{G}'_n$ based on Lemma 4.17, where we know that a graph G with a pair of false twins u and v is a PCG if and only if the induced subgraph $G[V(G) \setminus \{u\}]$ is a PCG.

Phase I-2: PCG Generator

We present a PCG generator to heuristically generate as many as possible PCGs with $n \geq 3$ vertices for which a given binary tree T is a witness tree and remove them from $\mathcal{G}'_n$. This PCG generator is based on two main ideas which are discussed below.

First, to avoid the repeated generation of the same PCGs, for a given binary tree T with n leaves, we assign edge-weights w by restricting the weights of some leaf-edges that can be mapped to each other under some tree automorphism. More precisely, for two leaf-edges e and e' incident with leaves u and u', respectively, such that there exists an automorphism that maps u' to u, we add the constraint $w(e') \leq w(e)$.

The second idea is to efficiently generate all PCGs with a fixed witness tree and weight assignment. By Calamoneri et al. [22], it is sufficient to consider positive integer weights w instead of real. Thus we have the following observation.

Observation 1. *For each* $\mathrm{PCG}(T, w, d_{\min}, d_{\max})$ *with positive integer valued* w*, it holds that* $\mathrm{PCG}(T, w, d_{\min}, d_{\max}) = \mathrm{PCG}(T, w, x - 0.5, y + 0.5)$ *such that* $x = \min\{d_{T,w}(a, b) \mid a, b \in L(T), d_{\min} \leq d_{T,w}(a, b) \leq d_{\max}\}$ *and* $y = \max\{d_{T,w}(a, b) \mid a, b \in L(T), d_{\min} \leq d_{T,w}(a, b) \leq d_{\max}\}$*.*

From Observation 1, it follows that for a binary tree T and a positive integer valued weight assignment w, it is sufficient to use only a finite number of pairs $(d_{\min}, d_{\max})$ to generate all PCGs with witness tree T and weight w as $\mathrm{PCG}(T, w, d_{\min}, d_{\max})$. Thus, in our PCG generator, based on Observation 1 we generate all PCGs with witness tree T and a positive integer weight assignment w by fixing x and y from the set $\{d_{T,w}(a, b) \mid a, b \in L(T)\}$.

For a tree $T \in \mathcal{T}_n$ and a leaf-edge $e = uv \in E(T)$ with leaf u, we define *equivalent edge class* $\mathrm{C}(e; T)$ to be the set of leaf-edges such that for each $u'v' \in \mathrm{C}(e; T)$ with leaf u' there exists an automorphism ψ of T such that $\psi(u') = u$ holds. We next give the main steps of our PCG generator to generate PCGs for each tree $T \in \mathcal{T}_n$:

(i) Randomly assign weights from a closed interval to the non-leaf-edges;

(ii) For each leaf-edge $e \in E(T)$ and a given subset $S \subseteq \mathrm{C}(e; T)$, randomly assign weight w in a closed interval to e such that $w(e') \leq w(e)$ holds for each $e' \in S$; and

(iii) For each pair of integers $x, y \in \{d_{T,w}(a, b) \mid a, b \in L(T)\}$ with $x \leq y$, construct the PCG graph $\mathrm{PCG}(T, w, x - 0.5, y + 0.5)$. Observe that the number of edges in $\mathrm{PCG}(T, w, x - 0.5, y + 0.5)$ is equal to the size of $\{a, b \in L(T) \mid d_{T,w}(a, b) \in [x, y]\}$, that is known before the generation of $\mathrm{PCG}(T, w, x - 0.5, y + 0.5)$. By using this observation, we do not generate PCGs of m edges if we know that we have already enumerated all PCGs with m edges in $\mathcal{G}'_n$ to avoid the generation of unnecessary PCGs.

We give an algorithmic description of our generator in Algorithm 3.

Algorithm 3 PCG Generator Based on Tree Symmetries

Input: A subset $\mathcal{G}' \subseteq \mathcal{G}_n$ with $n \geq 3$ vertices, a set $\mathcal{T}_n$ where the edge set of each tree in $\mathcal{T}_n$ is $\{e_1, e_2, \ldots, e_{2n-3}\}$ and $e_i, i \in [1, n]$ are the leaf-edges, a family of sets $\{S(e_i; T) \subseteq C(e_i; T) \mid i \in [1, n] \text{ and } T \in \mathcal{T}_n\}$, positive integers c, c', ℓ and ℓ', and an upper bound $\tau > 0$ on CPU time.

Output: Find PCGs from $\mathcal{G}'$ within an execution time $\leq \tau$.

```
1: Assume that each graph in $\mathcal{G}'$ has vertex set $\{1, 2, \ldots, n\}$;
2: $G[m] := \{G \in \mathcal{G}' \mid |E(G)| = m\}$ for all $m \in [0, \binom{n}{2})]$;
3: Candidate$[m] := |G[m]|$ for all $m \in [0, \binom{n}{2})]$;
4: **while** Elapsed time $< \tau$ **do**
5:  Randomly select weight $w_i \in [c, c'], i \in [n+1, 2n-3]$ of the non-leaf-edges;
6:  **for all** tree $T \in \mathcal{T}_n$ **do**
7:   Randomly select weight $w_i \in [\ell, \ell']$ of the leaf-edge $e_i$ such that $w_{i'} \leq w_i$
       for each $e_{i'} \in S(e; T)$;
8:   Compute $d_{T,w}(a, b)$ for all pairs $\{a, b\}$ with $1 \leq a < b \leq n$;
9:   Let $d_1 < d_2 < \cdots < d_k$ denote all distinct values in
       $\{d_{T,w}(a, b) \mid 1 \leq a < b \leq n\}$;
10:  $p_i := |\{\{a, b\} \mid 1 \leq a < b \leq n, d_{T,w}(a, b) = d_i\}|$ for each $i = 1, 2, \ldots, k$;
       /* $k \leq p_1 + p_2 + \cdots + p_k = \binom{n}{2}$ */
11:  **for each** pair $\{u, v\}$ with $1 \leq u < v \leq n$, let $[\{u, v\}] := i$ if $d_{T,w}(a, b) = d_i$;
12:  $q[0] := 0$;
13:  **for all** $i := 1, 2, \ldots, k$ **do**
14:   $q[i] := q[i-1] + p_i$;
15:   $d_{\min} := d_i - 1/2$;
16:   **for all** $j := i, i+1, \ldots, k$ **do**
17:    $d_{\max} := d_j + 1/2$;
18:    $m := q[j] - q[i-1]$; /* $m$ will be the number of edges in a PCG */
19:    **if** Candidate$[m] \geq 1$ **then**
20:     Construct PCG $G := (T, w, d_{\min}, d_{\max})$ by setting
          $V(G) := \{1, 2, \ldots, n\}$ and $E(G) := \{\{u, v\} \mid i \leq [\{u, v\}] \leq j\}$;
21:     **if** $G \in G[m]$ **then**
22:      $G[m] := G[m] \setminus \{G\}$; Candidate$[m] :=$ Candidate$[m] - 1$
23:     **end if**
24:    **end if**
25:   **end for**
26:  **end for**
27: **end for**
28: **end while**.
```

Phase I-3: Constructing Plausible Configuration

For a graph and a tree, in phase (I-3) we try to calculate an edge weight assignment and two reals bounded above by some value, such that the graph is isomorphic to the PCG due to the tree, weight assignment and interval bounded by reals. In other words, we try to construct a configuration with the given graph and tree such that the graph is a PCG due to the configuration, i.e., the configuration is plausible. More precisely, for a graph $G \in \mathcal{G}'_n$, a tree T and a real $\alpha \geq 0$, we construct a PCG$(T, w, d_{\min}, d_{\max})$ with $w, d_{\min}, d_{\max}$ bounded above by α, and try to confirm if there exist a bijection σ between vertex set of G and the leaf set of T and a 2-coloring λ over the set of non-adjacent pairs in G such that:

(i) For each edge in G it holds that the distance between their corresponding leaves under σ is in the interval $[d_{\min}, d_{\max}]$; and

(ii) The pairs of leaves corresponding to the pairs of non-adjacent vertices under σ with color 0 have distance strictly less than $d_{\min}$ and the others have distance strictly greater than $d_{\max}$.

To achieve this, for G, T, and α, we use the linear program, ILP$_{\text{suff}}(G, T, \alpha)$ discussed in Section 4.4. Recall that in addition to a weight w, real numbers $d_{\min}$ and $d_{\max}$ that are bounded above by α, ILP$_{\text{suff}}(G, T, \alpha)$ tries to find a mapping σ and a 2-coloring λ that satisfy (i) and (ii). By Lemma 4.18, the mapping σ is an isomorphism between G and PCG$(T, w, d_{\min}, d_{\max})$ if the objective value of ILP$_{\text{suff}}(G, T, \alpha)$ is 0, and hence we get a configuration (G, T, σ, λ) due to which G is a PCG, and therefore we remove G from the set $\mathcal{G}'_n$. However, if the objective value of ILP$_{\text{suff}}(G, T, \alpha)$ is greater than 0, then we cannot draw any conclusion as to whether G is a PCG or not, and therefore we further investigate G.

5.2.2 Phase II: Constructing Evidence Based on LP

In this section, we give details on phase (II), namely, how to prove that a graph is a PCG if it has not been so detected by the methods in phase (I), and constructing finite evidence if it is not a PCG by using essential configurations. This phase consists of three sub-phases: (II-1) Removing equivalent bijections; (II-2) Enumerating essential configurations; and (II-3) Solving configuration-based linear programs (LPs), which are explained in detail below.

Phase II-1: Removing Equivalent Bijections

Let $\mathcal{G}_n^*$ denote the set $\mathcal{G}_n'$ of graphs left after Phase I. For each pair of a graph $G \in \mathcal{G}_n^*$ and a tree $T \in \mathcal{T}_n$, this phase generates all bijections $\sigma \in \Sigma(G,T)$ and constructs a maximal set $\Sigma^*(G,T)$ by discarding bijections equivalent to one of the currently stored bijections in $\Sigma(G,T)$.

Recall that two bijections $\sigma_1, \sigma_2 \in \Sigma(G,T)$ are equivalent if there exists an automorphism π of T such that for vertices $u, v, p, q \in V(G)$ with $\sigma_1(u) = \pi(\sigma_2(p))$ and $\sigma_1(v) = \pi(\sigma_2(q))$ it holds that $uv \in E(G)$ if and only if $pq \in E(G)$. To test if two bijections are equivalent, for a graph G, tree T, and bijection σ, let $E_\sigma(G)$ is defined to be the set

$$E_\sigma(G) \triangleq \{\sigma(u)\sigma(v) \mid uv \in E(G)\}.$$

We define the *T-edge extended graph* $H(G,T,\sigma)$ to be the vertex colored graph obtained by adding the edges $\sigma(u)\sigma(v)$, for all $uv \in E(G)$, in T, i.e., $V(H(G,T,\sigma)) = V(T)$ and $E(H(G,T,\sigma)) = E(T) \cup E_\sigma(G)$, with a 2-vertex coloring π on $V(T)$ such that $\pi(v) = 0$ (resp., 1), if $v \in L(T)$ (resp., otherwise). An example of the graph $(H(G,T,\sigma),\pi)$ is given in Fig. 5.1. Thus for any $\sigma_1, \sigma_2 \in \Sigma(G,T)$, it holds that σ_1 is equivalent to σ_2 if and only if the graphs $(H(G,T,\sigma_1),\pi)$ and $(H(G,T,\sigma_2),\pi)$ are isomorphic.

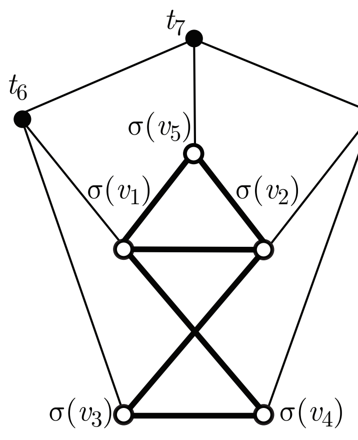


Figure 5.1. An example of the T-edge extended graph $(H(G,T,\sigma),\pi)$, where G, T and σ are given in Fig. 4.2. The edges in $E(T)$ and $E_\sigma(G)$ are illustrated by thin lines and thick lines, respectively. The 2-vertex coloring π is illustrated by representing values 0 and 1 by hallow and filled circles, respectively.

Phase II-2: Generating MIC4-free Configurations

We present a branch-and-bound algorithm that computes the set $\Lambda_{4\text{free}}$ of all 2-colorings $\lambda : \overline{E}(G) \to \{0,1\}$ such that (G,T,σ,λ) is MIC4-free for a graph $G \in \mathcal{G}_n$

with $n \geq 4$, a tree $T \in \mathcal{T}_n$ and a bijection and $\sigma \in \Sigma^*(G,T)$. We call such a 2-coloring λ an *MIC4-free coloring.* Since G, T, and σ are fixed, we assume for simplicity that $L(T) = V(G)$ in this subsection.

There are exactly $2^{|\overline{E}(G)|}$ 2-colorings $\lambda : \overline{E}(G) \to \{0,1\}$. We generate each of these 2-colorings by assigning a color from $\{0,1\}$ to each vertex pair in $\overline{E}(G)$, where we call an assignment $\lambda_Y : Y \to \{0,1\}$ of colors to a subset Y of $\overline{E}(G)$ a *partial coloring.* To search all the $2^{|\overline{E}(G)|}$ 2-colorings in a recursive fashion, we start with a partial coloring λ_Y with $Y := \emptyset$ such that none of the vertex pairs in $\overline{E}(G)$ is assigned a color from $\{0,1\}$; select a vertex pair $uv \in \overline{E}(G) \setminus Y$ to which no color is assigned yet; and generate two partial colorings $\lambda_{Y'}$ such that $Y' := Y \cup \{uv\}$ and $\lambda_{Y'}(ab) = \lambda_Y(ab)$, $ab \in Y$ by assigning color 0 or 1 to the pair uv.

We bound the branching process if the current partial coloring λ_Y cannot be extended to an MIC4-free coloring λ. That is, there exists a set $X \subseteq V(G)$ of four vertices such that for each pair $uv \in \binom{X}{2}$ either $uv \in E(G)$ or the value for the partial coloring $\lambda_Y(uv)$ is fixed, and a 3-coloring $\rho : \binom{X}{2} \to \{1,2,3\}$ such that $\rho(uv) = 2$ if $uv \in E(G)$ and $\rho(uv) = 2\lambda_Y(uv) + 1$ otherwise, where by Lemma 4.20 there does not exist an admissible pair with respect to ρ for the tree contraction $T\langle X \rangle$ of T.

To execute the branching operation systematically, we fix an arbitrary indexing $v_1, v_2, \ldots, v_n$ of the vertices in G. The aim of introducing the indexing is that given a vertex pair $v_iv_j \in \overline{E}(G)$, $i < j$, after assigning a color to the vertex pair v_iv_j, we want to uniquely select the next vertex pair to which no color is assigned yet by advancing the indices i and j while preserving $i < j$, that is, the next pair to be considered is either $v_{i+1}v_j$ if $1 \leq i < j-1$ or v_1v_{j+1} if $i = j-1$. Then, we are also able to uniquely determine the subset $S_{i,j} \subseteq \binom{\{v_1,v_2,\ldots,v_j\}}{2}$ of vertex pairs that have been already considered as follows:
(i) For each pair of indices p, q with $1 \leq p < q \leq j-1$, it holds that $v_pv_q \in S_{i,j}$; and
(ii) Index i is the smallest index such that $v_{i+1}v_j \notin S_{i,j}$,
where $Y = S_{i,j} \cap \overline{E}(G)$.
Observe that for any pair of indices i, j with $i < j$ the following recursion holds by the definition of the set $S_{i,j}$

$$S_{i,j} = \begin{cases} S_{i-1,j} \cup \{v_iv_j\} & \text{if } 1 < i \leq j-1, \\ S_{j-2,j-1} \cup \{v_1v_j\} & \text{if } i = 1. \end{cases} \tag{5.2.1}$$

To use Eq. (5.2.1) for a pair of indices i, j, $i < j$ and $(i,j) \neq (1,2)$, and the set $Y = S_{i,j} \cap \overline{E}(G)$, we store a partial coloring λ_Y as a set $F = \{(p, q; \lambda_Y(v_pv_q)) \mid$

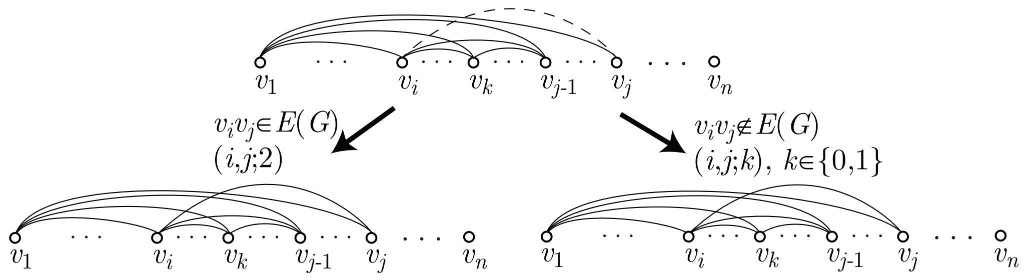


Figure 5.2. An illustration of the recursive relation given in Eq. (5.2.2) for an internal node $\Psi(F, i, j)$ when $i < j-1$. The dashed line between v_i and v_j shows that the pair (i, j) is being considered. A partial coloring is stored in the set F for each vertex pair $v_pv_q \in S_{i,j}$, indicated by solid lines connecting pairs of vertices.

$v_pv_q \in Y\} \cup \{(p, q; 2) \mid v_pv_q \in E(G)\}$ of triplets, and define the set

$$\Psi(F, i, j) \triangleq \{\lambda \in \Lambda_{\text{4free}} \mid (p, q; \lambda(v_pv_q)) \in F, \ \ \forall v_pv_q \in S_{i,j} \cap \overline{E}(G)\}$$

of all possible MIC4-free colorings that can be extended from the partial coloring λ_Y stored in F.

Then, we propose a branch-and-bound algorithm to generate the set Λ_{4free}. Note that $\Psi(F, i, j)$ defines one *node* in our branch-and-bound algorithm.

- Root node: We let $F = \{(i, j; 2) \mid v_iv_j \in E(G)\}$ and it holds that $\Lambda_{\text{4free}} = \Psi(F, 1, 2)$;

- Internal node: For any $1 \le i < j \le n$, $(i, j) \ne (1, 2)$ and a set F of triplets that stores a partial coloring λ_Y over the set $Y = S_{i,j} \cap \overline{E}(G)$, $\Psi(F, i, j)$ is an internal node. Observe that we have the following relations:

$$\Psi(F, i, j) = \begin{cases} \Psi(F, i+1, j) & \text{if } i < j-1 \text{ and } v_{i+1}v_j \in E(G), \\ \Psi(F, 1, j+1) & \text{if } i = j-1 \text{ and } v_1v_{j+1} \in E(G), \\ \bigcup_{\ell=0,1} \Psi(F \cup \{(i, j; \ell)\}, i+1, j) & \text{if } i < j-1 \text{ and } v_{i+1}v_j \in \overline{E}(G), \\ \bigcup_{\ell=0,1} \Psi(F \cup \{(i, j; \ell)\}, 1, j+1) & \text{if } i = j-1 \text{ and } v_1v_{j+1} \in \overline{E}(G). \end{cases} \tag{5.2.2}$$

  It is evident from Eq. (5.2.2) that the set F which stores a partial coloring is maintained during the recursive branching process. An illustration of the recursion when $i < j-1$ is given in Fig. 5.2.

- Bounding operation: An internal node $\Psi(F, i, j)$ with $j \ge 4$ is pruned if the current partial coloring cannot be extended to an MIC4-free coloring, i.e., $\Psi(F, i, j) = \emptyset$. To perform the bounding operation, for each subset

$\{v_h, v_k, v_i, v_j\} \subseteq V(G)$ with $1 \le h < k \le i-1$ we select four vertices $a, b, c,$ and d such that $\{a, b, c, d\} = \{v_h, v_k, v_i, v_j\}$ and the pairs of vertices $\{a, b\}$ and $\{c, d\}$ are siblings in the tree contraction $T\langle\{a, b, c, d\}\rangle$, i.e., it holds that

$$\delta_T(a, b) + \delta_T(c, d) < \delta_T(a, c) + \delta_T(b, d).$$

Then we check the existence of an admissible pair for the 3-coloring $\rho : \binom{\{a,b,c,d\}}{2} \to \{1, 2, 3\}$ such that for each $\{v_r, v_s\} \in \binom{\{a,b,c,d\}}{2}$ with $(r, s; k) \in F$ it holds that $\rho(v_r v_s) = k$ if $v_r v_s \in E(G)$, and $\rho(v_r v_s) = 2k+1$ otherwise. By Lemma 4.20, if such a 3-coloring ρ satisfies any of Eqs. (4.5.22) - (4.5.24), then there does not exist an admissible pair with respect to ρ for the tree contraction $T\langle\{a, b, c, d\}\rangle$ of T, and hence such an internal node $\Psi(F, i, j)$ is pruned.

- Leaf node: When $(i, j) = (1, n+1)$, then the set F stores a coloring $\lambda_{\overline{E(G)}}$, and we are at a leaf node.

Finally, from Λ_{4free}, we get all MIC4-free configurations with $G, T,$ and σ. An algorithmic overview of this method is given in Algorithms 4–6. The input of Algorithm 4 is a graph G, a tree T and a bijection σ, and the output is the set of all MIC4-free colorings. In this algorithm, we initialize a coloring λ which is then extended in Algorithm 5 with the recursive procedure explained in Eq 5.2.2. Finally, in Algorithm 6, we perform the bounding operation to check if the current coloring is MIC4-free coloring or not.

Algorithm 4 Generating MIC4-free Edge-colorings

Input: /* Global information: $G \in \mathcal{G}_n^*$ with vertex set $V(G) = \{v_1, v_2, \ldots, v_n\}$, $T \in \mathcal{T}_n$, $\sigma \in \Sigma^*(G, T)$, edge-distance δ_T between leaves in T, the current partial edge-coloring $\lambda : \{ij \mid 1 \le i < j \le n\} \to \{0, 1, 2, 3\}$, where $\lambda(ij)$ means $\lambda(v_i v_j)$ for $v_i v_j \in \binom{V(G)}{2}$ and the current set $\Lambda_{\text{mic4free}}$ of mic4free edge-colorings */

Output: The set of all MIC4-free edge-colorings in $\Lambda(G, T, \sigma)$ (possibly containing equivalent edge-colorings).

```
1: $\Lambda_{\text{mic4free}} := \emptyset$;
2: for all $i = 1, 2, \ldots, n-1$ do
3:     for all $j = i+1, i+2, \ldots, n$ do
4:         $\delta_T[ij] := |E_T(\sigma(v_i), \sigma(v_j))|$;
5:         if $v_i v_j \in E(G)$ then
6:             $\lambda[ij] := 2$
7:         else
8:             $\lambda[ij] := 0$
```

```
 9:         end if
10:     end for
11: end for;
12: GENERATE$(1, 2)$;
13: Output $\Lambda_{\mathrm{mic4free}}$.
```

Algorithm 5 Generating Extended MIC4-free Edge-colorings GENERATE(j, k)

Input: /* Global information: $G \in \mathcal{G}_n^*$ with vertex set $V(G) = \{v_1, v_2, \ldots, v_n\}$, edge-distance δ_T between leaves in T, the current partial edge-coloring $\lambda : \{ij \mid 1 \leq i < j \leq n\} \to \{0, 1, 2, 3\}$, and the current set $\Lambda_{\mathrm{mic4free}}$ of mic4free edge-colorings */
Integers j and k with $1 \leq j < k \leq n$.

Output: The set of all mic4free edge-colorings $\lambda' \in \Lambda(G, T, \sigma)$ that can be extended from the current partial edge-coloring λ.

```
 1: if $j = n - 1$ and $k = n$ then
 2:     $\Lambda_{\mathrm{mic4free}} := \Lambda_{\mathrm{mic4free}} \cup \{\lambda\}$ for the current edge-coloring $\lambda$; return
 3: else
 4:     if $j \leq k - 2$ then
 5:         $j := j + 1$
 6:     else/* $j = k - 1$ and $k \leq n - 1$ */
 7:         $j := 1$; $k := k + 1$
 8:     end if;
 9:     if $\lambda(jk) = 2$ then
10:         if PLAUSIBILITY$(j, k)$ = mic4free then
11:             GENERATE$(j, k)$
12:         end if
13:     else
14:         for all color a $\in \{1, 3\}$ do
15:             $\lambda(jk) :=$ a;
16:             if PLAUSIBILITY$(j, k)$ = mic4free then
17:                 GENERATE$(j, k)$
18:             end if
19:         end for
20:     end if
21: end if.
```

Algorithm 6 Plausibility Test PLAUSIBILITY(j, k)

Input: /* Global information: $G \in \mathcal{G}_n^*$ with vertex set $V(G) = \{v_1, v_2, \ldots, v_n\}$, edge-distance δ_T between leaves in T, the current partial edge-coloring λ :

$\{ij \mid 1 \leq i < j \leq n\} \to \{0,1,2,3\}$ */
Integers j and k with $1 \leq j < k \leq n$.

Output: Message `mic4free` if $k \leq 3$ or $G[v_h, v_i, v_j, v_k]$ is MIC4-free for all h and i with $1 \leq h < i \leq j-1$; and message `implausible` otherwise.

```
1:  $A :=$ mic4free;
2:  **if** $k \geq 4$ **then**
3:      **for all** $h := 1, 2, \ldots, j-2$ **do**
4:          **for all** $i := h+1, h+2, \ldots, j-1$ **do**
5:              $X := \{h, i\}$; $Y := \{j, k\}$;
6:              **if** $\delta_T(hj) + \delta_T(ik) < \delta_T(hi) + \delta_T(jk)$ **then**
7:                  $X := \{h, j\}$; $Y := \{i, k\}$
8:              **end if**;
9:              **if** $\delta_T(hk) + \delta_T(ij) < \delta_T(hi) + \delta_T(jk)$ **then**
10:                 $X := \{h, k\}$; $Y := \{i, j\}$
11:             **end if**;
12:             Let $i_1, i_2, i_3, i_4$ denote the indices $X = \{i_1, i_2\}$ and $Y = \{i_3, i_4\}$
                  (choose any of the $2 \times 2 \times 2$ combinations);
13:             **if** $|\lambda(i_1i_2) - \lambda(i_3i_4)| \leq 1$ **then**
14:                 **if** $|\lambda(i_1i_3) - \lambda(i_2i_4)| \leq 1$ and
                          $\lambda(i_1i_2) + \lambda(i_3i_4) - 2 \geq \lambda(i_1i_3) + \lambda(i_2i_4)$ **then**
15:                     $A :=$ implausible
16:                 **end if**;
17:                 **if** $|\lambda(i_1i_4) - \lambda(i_2i_3)| \leq 1$ and
                          $\lambda(i_1i_2) + \lambda(i_3i_4) - 2 \geq \lambda(i_1i_4) + \lambda(i_2i_3)$ **then**
18:                     $A :=$ implausible
19:                 **end if**
20:             **end if**;
21:             **if** $|\lambda(i_1i_3) - \lambda(i_2i_4)| \leq 1$, $|\lambda(i_1i_4) - \lambda(i_2i_3)| \leq 1$ and
                    $|(\lambda(i_1i_3) + \lambda(i_2i_4)) - (\lambda(i_1i_4) + \lambda(i_2i_3))| \geq 2$ **then**
22:                 $A :=$ implausible
23:             **end if**
24:         **end for**
25:     **end for**
26: **end if**;
27: Output $A$.
```

Phase II-3: Configuration-based LPs

We use linear programming formulations discussed in Section 4.3 to handle the difficulty of the infinite search space, and the construction of finite size evidence. For an MIC4-free configuration (G, T, σ, λ), obtained in phase (II-2), we solve the formulation $\mathrm{LP}(G, T, \sigma, \lambda)$ given in Section 4.3. The formulation has a solution w, $d_{\min}$ and $d_{\max}$ if and only if the configuration is plausible. Note that instead of trying all possible real weights to confirm if a graph is a PCG or not due to the given configuration, it suffices to solve this single formulation. Finally, to get a definite proof for the implausibility of a configuration (G, T, σ, λ), we use the formulation, $\mathrm{DLP}(G, T, \sigma, \lambda)$ introduced in Section 4.3 . Recall that the linear program $\mathrm{DLP}(G, T, \sigma, \lambda)$ admits a solution if and only if $\mathrm{LP}(G, T, \sigma, \lambda)$ does not, by Theorem 4.6. Hence, the solution of $\mathrm{DLP}(G, T, \sigma, \lambda)$ will serve as a finite size evidence to show that G is not a PCG due to (G, T, σ, λ).

5.2.3 Experimental Results and Comparison for Eight Vertices

We present the computational results of the proposed method to enumerate all PCGs with eight vertices. To show the computation efficiency of our proposed method, we also enumerated PCGs with eight vertices by method due to Azam et al. [3]. We provide a computation comparison of the method due to Azam et al. [3] and our proposed method for enumerating PCGs with eight vertices. To avoid any numerical errors in our experiments, we solved LP and DLP as ILPs by using the `IBM ILOG CPLEX 12.8` solver. There are 12346 graphs with eight vertices, i.e., $|\mathcal{G}_8| = 12346$. We store graphs in a canonical form for easy comparison between graphs. Canonical forms were obtained by the `NAUTY` [47] software. There are four trees T_1, T_2, T_3 and T_4 in $\mathcal{T}_8$. The edge sets of these trees are listed in Table 5.1 by representing an edge jk as j-k, where we take $V(T_i) = \{1, 2, \ldots, 14\}$, $i = 1, 2, 3, 4$. In the rest of this section, we first discuss the experimental results of our proposed method.

Table 5.1. Edge sets of the four trees in $\mathcal{T}_8$ with vertex set $\{1, 2, \ldots, 14\}$

i	Edge sets of the tree T_i
1	1-9, 2-9, 3-10, 4-11, 5-12, 6-13, 7-14, 8-14, 9-10, 10-11, 11-12, 12-13, 13-14
2	1-9, 2-9, 3-10, 4-10, 5-11, 6-11, 7-12, 8-12, 9-13, 10-13, 11-14, 12-14, 13-14
3	1-9, 2-9, 3-10, 4-10, 5-12, 6-13, 7-14, 8-14, 9-11, 10-11, 11-12, 12-13, 13-14
4	1-9, 2-9, 3-11, 4-10, 5-10, 6-12, 7-14, 8-14, 9-11, 10-12, 11-13, 12-13, 13-14

Graph Screening Phase

Let $\mathcal{G}' := \mathcal{G}_8$. The graph screening phase was executed on a PC with Intel Core i7-500 processor, running at 2.70GHz, 16GB of memory and Windows 10. Note that in this phase we only removed some PCGs, since all graphs with at most seven vertices are known to be PCGs [20]. By removing the graphs with false twins from the set $\mathcal{G}'$, we are left with 8047 graphs. From these graphs we removed the non-biconnected graphs. After this step, we have 4959 remaining graphs. The running time of each of these steps is less than one minute. Then, from these graphs we eliminated 4846 PCGs in 3.5 hours by using the PCG generator by randomly selecting weights for edges from the interval $[1, 50]$, i.e., we are left with 13 graphs for each of which and each tree in $\mathcal{T}_8$ we solved ILP$_{\text{suff}}$ by setting $\alpha = 300$. The maximum, average, and minimum execution times of ILP$_{\text{suff}}$ in this experiment are 123, 28 and 1 second, respectively. By this step, six out of 13 graphs are identified as PCGs. Thus, by the application of the graph screening phase on $\mathcal{G}'$, 12339 graphs out of 12346 graphs are identified as PCGs, i.e., $|\mathcal{G}_8^*| = 7$.

Constructing Evidence Based on LP

We computed non-equivalent bijections and MIC4-free configurations on a PC with Intel(R) Xeon(R) E5-1600v3 processor, running at 3.00GHz, 64GB of memory and Windows 7. For a graph $G \in \mathcal{G}_8^*$ and a tree $T \in \mathcal{T}_8$, the maximum, average, and minimum computation times to compute the set $\Sigma^*(G, T)$ are 619, 515, and 490 seconds, respectively.

After finding non-equivalent bijections for each of the graphs in $\mathcal{G}_8^*$, we applied the proposed branch-and-bound algorithm to generate all MIC4-free colorings with graphs in $\mathcal{G}_8^*$. There are three graphs in $\mathcal{G}_8^*$ which have no MIC4-free colorings, and therefore they are NPCGs. The remaining four graphs have two, five, eight, and nine MIC4-free colorings. For each $G \in \mathcal{G}_8^*$ and $T \in \mathcal{T}_8$ the proposed branch-and-bound algorithm computed all MIC4-free colorings in less than one second.

For each MIC4-free configuration with a graph in $\mathcal{G}_8^*$, ILP and DILP are solved. The solver solved these ILPs for all MIC4-free configurations with a graph in $\mathcal{G}_8^*$ in less than one second. The solutions to DILPs are given at
`http://www-or.amp.i.kyoto-u.ac.jp/~azam/8computational_evidence.zip`.
The experimental results reveal that all of the graphs in $\mathcal{G}_8^*$ are NPCGs. Note that these seven NPCGs are MNPCGs, since all graphs with at most seven vertices are PCGs [20]. These seven NPCGs are illustrated in Fig 5.3, where the graphs in Fig 5.3(a) and (b) have already been shown to be NPCGs in [10] and [27], respec-

tively, while the remaining are the newly discovered MNPCGs. We summarize these findings in the following theorem.

Theorem 5.7. *There are exactly seven graphs with eight vertices which are NPCGs.*

A proof of Theorem 5.7 can be deduced form the solutions to DILPs for MIC4-free configurations with graphs from $\mathcal{G}_8^*$.

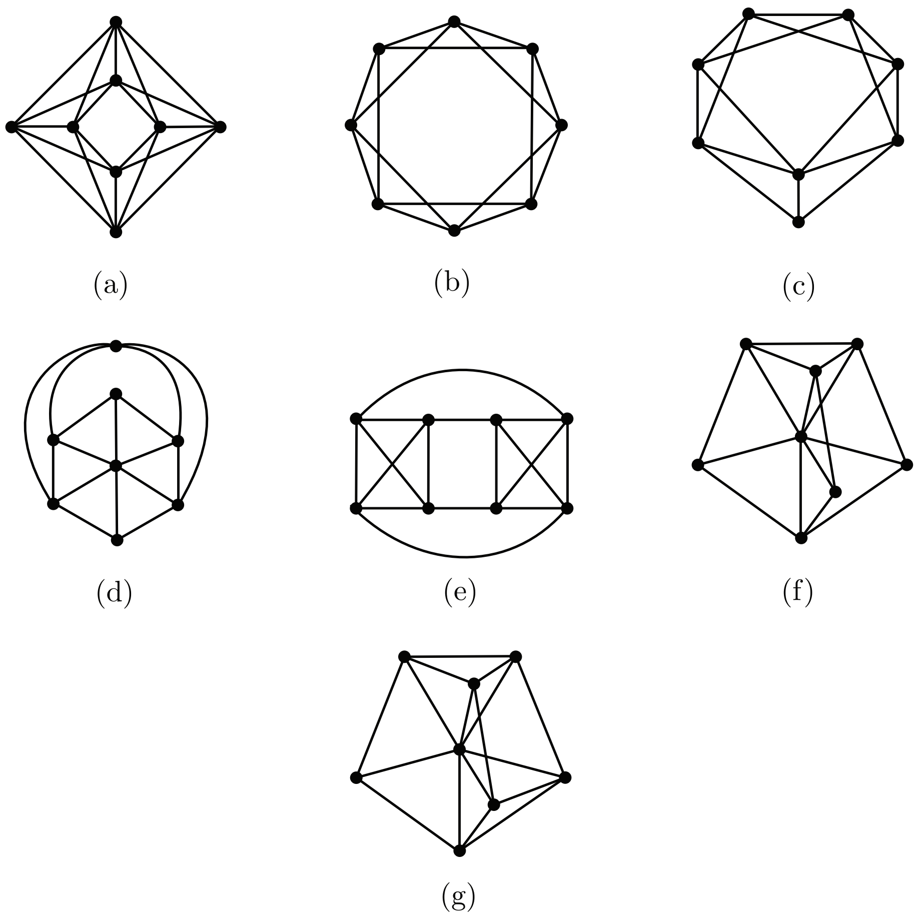


Figure 5.3. The seven NPCGs with eight vertices: (a) The NPCG given in [10]; (b) The NPCG given in [27]; and (c)-(g) The five NPCGs discovered by the proposed method.

Next we present a comparison of the computational results of the method due to Azam et al. [3] and the proposed method.

Computational Comparison

We compared the computational efficiency of each step of the method due to Azam et al. [3] and our improved method for enumerating PCGs with eight

Table 5.2. Computation comparison between the enumeration method by Azam et al. [3] and the proposed method for eight vertices

Steps	# graphs left [Time (min.)]			
	The Method by Azam et al. [3]		Proposed Method	
Number of graphs with eight vertices	12346			
Removing graphs with false twins	12346	[N.A.]	8047	[< 1]
Removing non-biconnected graphs	7123	[< 1]	4959	[< 1]
Using PCG generator	7	[460]	13	[210]
Using $\mathrm{ILP}_{\mathrm{suff}}$	7	[N.A.]	7	[12]
Subtotal for graph screening	7	[460]	7	[222]
Computing non-equivalent bijections	[231]			
PCG/NPCG evidence construction		[32229]		[< 1]
Proven MNPCG	7			
Total time		[32920]		[453]

N.A. denotes that the corresponding step is not included in the method.

vertices in Table 5.2. For each step, we list the output in terms of graphs and computation time in minutes in Table 5.2. The experiments for both methods are performed in the same computation environment. For the steps which are not available in the method due Azam et al. [3], we show the output of the previous step and write N.A. for computation time. Both methods took less than one minute to remove non-biconnected graphs. Seven and 13 graphs left after using PCG generator for 460 and 210 minutes for the method due Azam et al. [3] and proposed method, respectively. Note that after the graph screening phase both methods obtained a set $\mathcal{G}_8^*$ with seven graphs. The method due to Azam et al. [3] took 460 minutes to obtain a set $\mathcal{G}_8^*$, while the proposed method took 224 minutes to obtain a set $\mathcal{G}_8^*$. The step of computing non-equivalent bijections for each graph in $\mathcal{G}_8^*$ is common in both methods and took 231 minutes for each method. The method due to Azam et al. [3] and the proposed method took 32229 minutes and less than 1 minute, respectively, to construct evidence whether each of the graphs in $\mathcal{G}_8^*$ is PCGs or not. Both methods proved that all graphs in $\mathcal{G}_8^*$ are NPCGs. The method due to Azam et al. [3] and the proposed method took 32920 and 453 minutes, respectively, to enumerate all PCGs with eight vertices. Hence it is evident from the computational results that the proposed method is efficient as compared to the method due to Azam et al. [3].

5.3 Enumerating Feasible Pairs

Recall that phase (II-1) in Section 5.2.2 involves computation of equivalent bijection. However, the number of bijections increases exponentially with the increase in the number n of vertices in the underlying graphs. Therefore we propose a new branch-and-bound algorithm that for a given graph generates all pairs of trees and 2-colorings that correspond to MIC4-free configurations with a fixed bijection, since we can fix a bijection without loss of generality due to the vertex labeling of the graph and tree. We call such a pair of tree and 2-coloring a *feasible* pair. We later replace phases (II-1) and (II-2) with this new algorithm.

Following is an intuitive description of our branch-and-bound algorithm. We examine all pairs in a recursive manner, starting with a binary tree with two leaves and a 2-coloring where no color has been assigned to non-adjacent pairs. If each of the non-adjacent pairs that are leaves in the current tree have been assigned a color, then we extend the current tree by adding a new vertex z subdividing each edge xy, together with a new leaf ℓ adjacent to z. Otherwise, we choose one of the non-adjacent pairs that has not been assigned a color, and extend the current 2-coloring into two colorings that have color 0 or 1 for this uncolored pair. We name the above operations the *tree extension operation* and the *coloring extension operation*, respectively.

If the current pair of tree and 2-coloring cannot be extended to a feasible pair, then we bound the current branch . Recall that a necessary condition for a current pair of a tree and a 2-coloring to be extendable is given by the presence of an MIC4 subconfiguration. To efficiently test whether an MIC4 subconfiguration exists, we use a characterization given in Lemma 4.20.

To perform our algorithm systematically, we fix the vertex set of a graph G with n vertices to be $\{v_1, v_2, \ldots, v_n\}$. Then, fixing the identity bijection, we generate feasible pairs that correspond to MIC4-free configurations. As the root of our branching procedure, we start with $X = \{v_1, v_2\}$, the unique tree with a single edge $\{v_1 v_2\}$ and a 2-coloring that is currently not defined on any pair of non-adjacent vertices that are in X. Next, we discuss the tree extension and the coloring extension operations in more detail below.

Tree extension operation
Assume that all pairs of non-adjacent vertices that are in X with $|X| = k$ are colored. Then we perform the tree extension operation for each edge xy in the current tree in increasing order of the label of y, by adding a new degree-three vertex $z = v_{n+k-2+1}$ that subdivides the edge xy, and a new leaf $\ell = v_{k+1}$ incident to z. Note that at any stage of the algorithm it holds that $X = \{v_1, v_2, \ldots, v_k\}$

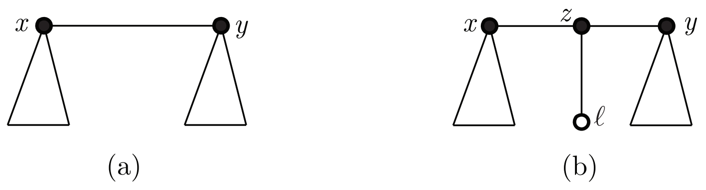


Figure 5.4. An illustration of the tree extension operation: (a) A tree T; and (b) The extended tree T' of T obtained by adding a new vertex z and a new leaf ℓ by subdividing edge $xy \in E(T)$.

for some $k \in [2, n]$. As a result of the tree extension operation on the current tree, we get a unique tree with leaf set $X \cup \{v_{k+1}\}$. We call such a tree an *extended tree* of the current tree. An illustration of the tree extension operation is given in Fig. 5.4. With the following lemma, we argue that our branching procedure generates all trees that have the leaf set $V(G)$.

Lemma 5.21. *For any binary tree T with $|L(T)| \geq 3$ there exists a tree H with $L(H) \subsetneq L(T)$ and $|L(T)| = |L(H)| + 1$ such that T is an extended tree of H.*

Proof. Let $z\ell \in E(T)$ be a leaf-edge such that z and ℓ are non-leaf and leaf vertices, respectively. There always exists such an edge $z\ell$ since $|L(T)| \geq 3$. Let x and y denote the neighbors of z other than ℓ. Then by applying the tree extension operation on the tree H such that $V(H) = V(T) \setminus \{z, \ell\}$ and $E(H) = (E(T) \setminus \{z\ell, yz, xz\}) \cup \{xy\}$, we get T, from which the claim follows. □

Coloring extension operation

Notice that, for an integer $k \in [2, n]$, due to the systematic selection of vertices it follows that the set of uncolored pairs are $v_j v_k$, $j \leq k - 1$. After assigning a color to the vertex pair $v_j v_k, j < k$, we consider the next pair uv to color to be the pair $v_{j+1} v_k$ if $j < k - 1$ and $v_1 v_{k+1}$ otherwise (if $j = k - 1$). For any pair j, k with $j < k$, we define $A_{j,k}$ to be the set consisting of all those pairs that are already considered for coloring. Thus it holds that $v_c v_d \in A_{j,k}$ for all c, d such that $c < d \leq k - 1$, and j is the smallest integer for which we have $v_{j+1} v_k \in A_{j,k}$. This implies that for any pair j, k with $j < k$, we can uniquely determine the set of colored vertex pairs as the set of the pairs of non-adjacent vertices that are in $A_{j,k}$.

Subproblem and recursion

Let $k \in [2, n]$ be an integer. We formalize a subproblem and recursive relations for our algorithm to enumerate MIC4-free configurations. For a pair of integers j and k with $j < k$ and the set $Y = \overline{E}(G) \cap A_{j,k}$ of colored pairs, we represent a 2-coloring $\lambda : Y \to \{0, 1\}$ by a set $C = \{(c, d; \lambda(v_c v_d)) \mid v_c v_d \in Y\} \cup \{(c, d; 2) \mid$

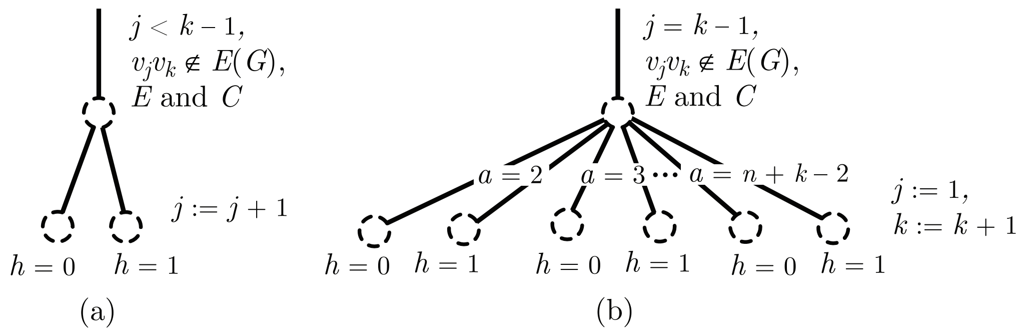


Figure 5.5. An illustration of the branching procedure for a subproblem $\mathcal{S}(E; C, j, k)$ with $v_jv_k \in \overline{E}(G)$, edge set E and coloring C: (a) Two branches obtained when $j < k - 1$ by applying the coloring extension operation; and (b) The branches obtained when $j = k - 1$ by applying the tree extension and coloring extension operations for each edge $xv_a \in E, a \in [2, k] \cup [n + 1, n + k - 2]$.

$v_cv_d \in E(G)\}$). Thus, we represent a pair (T, λ) of a tree T with edge set E and a current coloring λ by the pair (E, C) of sets. Finally, we define a subproblem $\mathcal{S}(E; C, i, j)$ to be the set of all possible feasible pairs that can be extended from the current pair (E, C). Observe that $\mathcal{S}(\{v_1, v_2\}; \emptyset, 1, 2)$ is the required set of all feasible pairs of trees and 2-colorings. Furthermore, for $j < k$ with $k \in [3, n]$, $z = v_{n+k+2-1}$ and $\ell = v_{k+1}$, we have the following recursion
$\mathcal{S}(E; C, j, k) =$

$$\begin{cases} \mathcal{S}(E; C, j+1, k), & j<k-1, v_jv_k \in E(G), \\ \bigcup\limits_{xv_a \in E} \mathcal{S}((E \setminus \{xv_a\}) \cup \{xz, v_az, \ell z\}; C, 1, k+1), & j=k-1, v_jv_k \in E(G), \\ \bigcup\limits_{h=0,1} \mathcal{S}(E; C \cup \{(j,k;h)\}, j+1, k), & j<k-1, v_jv_k \in \overline{E}(G), \\ \bigcup\limits_{\substack{xv_a \in E, \\ h=0,1}} \mathcal{S}((E \setminus \{xv_a\}) \cup \{xz, v_az, \ell z\}; C \cup \{(j,k;h)\}, 1, k+1), & j=k-1, v_jv_k \in \overline{E}(G). \end{cases} \tag{5.3.3}$$

An illustration of the recursion for the case when $v_jv_k \in \overline{E}(G)$ in Eq. (5.3.3) is given in Fig. 5.5(a)-(b).

Bounding procedure

Let $k \geq 4$ be an integer. We bound a subproblem $\mathcal{S}(P; C, j, k)$ if the current pair of tree T and coloring λ cannot be extended to a feasible pair. To perform this bounding operation, we need to check if there exists a subset $Z \subseteq \{v_1, v_2, \ldots, v_k\}$ of size 4 such that for each pair of non-adjacent vertices that are in Z, the current coloring λ is defined and the configuration induced by Z is implausible. To efficiently verify this, we use a characterization of MIC4 given in Lemma 4.20. However, to use this method, for each subset $Z = \{v_h, v_k, v_i, v_j\}$ such that $h < i \leq j-1$, we need to find two pairs of vertices in Z such that each pair has a common neighbor in the tree contraction $T\langle Z\rangle$. For this purpose we use Lemma 5.22.

Lemma 5.22. *Let T be a tree with $|L(T)| \geq 4$ and $Z \subsetneq L(T)$ be with $|Z| = 4$. Let $a_1, b_1, a_2, b_2 \in Z$ be such that the vertex sets of $P_T(a_1, b_1)$ and $P_T(a_2, b_2)$ are disjoint. Then for each $i = 1, 2$, the vertices a_i and b_i have a common neighbor in $T\langle Z\rangle$.*

Proof. Since $P_T(a_1, b_1)$ and $P_T(a_2, b_2)$ are disjoint, there exist two unique non-leaf vertices $c, d \in V(T)$ such that $c \in V(P_T(a_1, b_1))$, $d \in V(P_T(a_2, b_2))$ and $V(P_T(c, d)) \subsetneq V(P_T(a_1, b_2))$. Then to obtain the tree contraction $T\langle Z\rangle$, we first remove those vertices in T that are not contained in any path connecting two vertices in Z. That is, we recursively remove all leaves from $L(T) \setminus Z$. This implies that all non-leaf vertices in $V(T)$ except c and d are of degree 2 after the removal of all leaves. Thus, by removing all those vertices of degree 2 to get $T\langle Z\rangle$, we can see that c is the common neighbor of a_1, b_1 and d is the common neighbor of a_2, b_2, from which the claim follows. □

We give an algorithmic description of our branch-and-bound algorithm in Algorithms 7-12. In these algorithms, for an integer $k \in [2, n]$, $X = \{v_1, v_2, \ldots, v_k\}$ and $T \in \mathcal{T}(X)$, we use

- n_{inner}: the number of internal vertices in the current partial tree T;
- $p[]$: $p[i]$ stores the index j of the parent v_j of a vertex $v_i \in X \setminus \{v_1\}$ in the current tree T;
- lca$[,]$: lca$[i, j]$ stores the index h of the least common ancestor v_h of two vertices $v_i, v_j \in X$ in the current rooted tree T, where lca$[i, j] = 0$ if the least common ancestor of vertices $v_i, v_j \in V(G)$ has not been computed;
- visit$[]$: visit$[i]$ stores the index i of a vertex v_i, where we initialize visit$[i] := 0$, $1 \leq i \leq 2n-2$;
- $\lambda[,]$: λ is a function $\{(i, j) \mid 1 \leq i < j \leq n\} \to \{0, 1, 2, 3\}$, where $\lambda[i, j] = 2$ means that $v_iv_j \in E(G)$ and $\lambda[i, j] = 0$ means that $v_iv_j \notin E(G)$ and no color is assigned to v_iv_j, $\lambda[i, j] = 1$ (resp., 3) means that $v_iv_j \notin E(G)$ and $d_{T,w}(v_i, v_j) < d_{\min}$ (resp., $d_{T,w}(v_i, v_j) > d_{\max}$) holds.

The input of Algorithm 7 is a graph G with n vertices and the output of Algorithm 7 is the set Γ of all feasible pairs (T, λ). In this algorithm we initialize visit$[i]$, $i \in [1, n-1]$ and lca$[i, j]$, $i \in [1, n-1]$ and $j \in [i+1, n]$ that will be used in Algorithm 10, 11 and 12 to test if two paths are disjoint or intersecting. Then, we initialize a 4-coloring $\lambda[i, j]$, $i \in [1, n-1]$ and $j \in [i+1, n]$ such that $\lambda[i, j] = 2$ if $v_i v_j \in E(G)$ and $\lambda[i, j] = 0$ otherwise. We next initialize the current set Γ and current partial tree with two vertices v_1 and v_2. Finally, we perform the tree extension and coloring extension operations on the current partial tree and coloring.

Algorithm 7 Generating Feasible Pairs

```
Input: /* Global information: $G = (V(G) = \{v_1, v_2, \ldots, v_n\}, E(G)) \in \mathcal{G}_n$,
    $n_{\text{inner}}$, the current tree $p[] = T_k \in \mathcal{T}(\{v_1, \ldots, v_k\})$, the current partial edge-
    coloring $\lambda[,]$, and the current set $\Gamma$ of feasible pairs. */
Output: The set of all feasible pairs for $G$.
 1: for all $i = 1, 2, \ldots, n-1$ do
 2:     visit$[i] := i$;
 3:     for all $j = i+1, i+2, \ldots, n$ do
 4:         lca$[i, j] := 0$;
 5:         if $v_i v_j \in E(G)$ then
 6:             $\lambda[i, j] := 2$
 7:         else
 8:             $\lambda[i, j] := 0$
 9:         end if
10:     end for
11: end for;
12: $\Gamma := \emptyset$; $p[2] := 1$; $n_{\text{inner}} := 0$;
13: GENERATE$(1, 2, 0)$;
14: Output $\Gamma$.
```

In Algorithm 8, we perform the tree extension and the coloring extension operations if the current pair of tree and coloring can be extended to a feasible pair. This algorithm also adds the newly generated feasible pair in the set Γ. From lines 1-3 and 28-37, Algorithm 8 performs the tree extension operation when $j = k-1$ followed by the coloring extension operation for the cases when $v_j v_k \in E(G)$ and otherwise. At line 4, we test if the current pair (p, λ) can be extended to a feasible pair or not by using Algorithm 8. From line 7-15, we get a 2-coloring $\lambda' : \overline{E}(G) \to \{0, 1\}$ such that (p, λ') is feasible. From lines 18-27, we perform coloring extension operation when $j < k-1$. Recall that in this case we only extend the coloring but not the current tree. From lines 20-21, the coloring extension operation is performed when $v_j v_k \in E(G)$, while from lines 22-26, the coloring extension operation is performed for $v_j v_k \notin E(G)$.

Algorithm 8 Generating Extended Pairs GENERATE($j, k,$ `add`)

Input: /* Global information: $G = (V(G) = \{v_1, v_2, \ldots, v_n\}, E(G)) \in \mathcal{G}_n$, n_{inner}, the current tree $p[] = T_k \in \mathcal{T}(\{v_1, \ldots, v_k\})$, the current partial edge-coloring $\lambda[,]$, and the current set Γ of feasible pairs. */
Integers j and k with $1 \leq j < k \leq n$, and an integer `add` $\in \{0\} \cup \{2, 3, \ldots, k-1\} \cup \{n+1, \ldots, n+n_{\text{inner}}\}$.

Output: If `add` ≥ 2 then a new inner vertex in the current tree is created as the parent of the vertex $v_{\texttt{add}}$. Then add a new edge color $c \in \{1, 3\}$ to the vertex pair $\{v_j, v_k\}$ (if $v_j v_k \notin E(G)$). Test whether there is an MIC4 subconfiguration containing $\{v_j, v_k\}$ or not. If a new MIC4 subconfiguration is created, then return. Otherwise return the set of all feasible pairs (T', λ') such that T' and λ' can be extended from the current subtree p and partial edge-coloring λ.

```
 1: if add $\geq 2$ /* Enlarge $T_k$ */ then
 2:     $n_{\text{inner}} := n_{\text{inner}} + 1$; $p[n + n_{\text{inner}}] := p[\texttt{add}]$; $p[\texttt{add}] := p[k] := n + n_{\text{inner}}$
 3: end if;
 4: if TEST$(j, k)$ = implausible then
 5:     return
 6: end if;
 7: if $j = n - 1$ and $k = n$ then
 8:     /* Getting a 2-coloring over $\{0, 1\}$ color set */
 9:     for all $i = 1, 2, \ldots, n-1$ do
10:         for all $\ell = i+1, i+2, \ldots, n$ do
11:             if $v_i v_\ell \in \overline{E}(G)$ then
12:                 $\lambda'[i, \ell] := (\lambda[i, \ell] - 1)/2$
13:             end if
14:         end for
15:     end for;
16:     $\Gamma := \Gamma \cup \{(p, \lambda')\}$ for the current tree $p$ and edge-coloring $\lambda$; return
17: else
18:     if $j \leq k - 2$ then
19:         $j := j + 1$;
20:         if $\lambda[j, k] = 2$ then
21:             GENERATE$(j, k, 0)$
22:         else
```

```
23:             for all color $\mathtt{c} \in \{1, 3\}$ do
24:                 $\lambda[j, k] := \mathtt{c}$; Generate$(j, k, 0)$
25:             end for
26:         end if
27:     else
28:         $j := 1$; $k := k + 1$;
29:         for each vertex $\mathtt{a} \in \{2, 3, \ldots, k-1\} \cup \{n+1, n+2, \ldots, n+n_{\text{inner}}\}$ do
30:             if $\lambda[j, k] = 2$ then
31:                 Generate$(j, k, \mathtt{a})$
32:             else
33:                 for all color $\mathtt{c} \in \{1, 3\}$ do
34:                     $\lambda[j, k] := \mathtt{c}$; Generate$(j, k, \mathtt{a})$
35:                 end for
36:             end if
37:         end for
38:     end if
39: end if.
```

In Algorithm 9, we test if the current pair of tree and coloring can be extended to a feasible pair or not by testing the inequalities in the characterization of MIC4 given in Lemma 4.20. In fact here we perform bounding operation of our branch-and-bound algorithm. Thus, from lines 3-13, the algorithm finds the vertex pairs that are leaves with common parents in the tree contraction of the current tree due to $\{v_h, v_i, v_j, v_k\}$. This is done by testing if the paths are disjoint or intersecting using Algorithm 10. From line 14-29, we test the inequalities given in Lemma 4.20.

Algorithm 9 Testing Plausibility Test(j, k)

Input: /* Global information: $G = (V(G) = \{v_1, v_2, \ldots, v_n\}, E(G))$, the current tree $p[] = T_k \in \mathcal{T}(\{v_1, \ldots, v_k\})$, and the current partial edge-coloring $\lambda[,]$. */ Integers j and k with $1 \leq j < k \leq n$.

Output: `mic4free` if $k \leq 3$ or $G[v_h, v_i, v_j, v_k]$ in the current tree T_k is MIC4-free for all h and i with $1 \leq h < i \leq j - 1$; `implausible` otherwise.

```
1: $A :=$ mic4free;
2: if $k \geq 4$ then
3:     for all $h := 1, 2, \ldots, j-2$ do
4:         for all $i := h+1, h+2, \ldots, j-1$ do
5:             if Disjoint$(h, i, j, k) =$ disjoint then
6:                 $X := \{h, i\}$; $Y := \{j, k\}$
7:             else
```

```
 8:             if $\textsc{Disjoint}(h,k,i,j) =$ disjoint then
 9:                 $X := \{h,k\}$; $Y := \{i,j\}$
10:             else
11:                 $X := \{h,j\}$; $Y := \{i,k\}$
12:             end if
13:         end if;
14:         Let $i_1, i_2, i_3, i_4$ denote the indices $X = \{i_1, i_2\}$ and $Y = \{i_3, i_4\}$;
                    /* choose any of the $2 \times 2 \times 2$ combinations */
15:         if $|\lambda[i_1,i_2] - \lambda[i_3,i_4]| \leq 1$ then
16:             if $|\lambda[i_1,i_3] - \lambda[i_2,i_4]| \leq 1$ and
                    $\lambda[i_1,i_2] + \lambda[i_3,i_4] - 2 \geq \lambda[i_1,i_3] + \lambda[i_2,i_4]$ then
17:                 $A :=$ implausible
18:             end if;
19:             if $|\lambda[i_1,i_4] - \lambda[i_2,i_3]| \leq 1$ and
                    $\lambda[i_1,i_2] + \lambda[i_3,i_4] - 2 \geq \lambda[i_1,i_4] + \lambda[i_2,i_3]$ then
20:                 $A :=$ implausible
21:             end if
22:         end if;
23:         if $|\lambda[i_1,i_3] - \lambda[i_2,i_4]| \leq 1$, $|\lambda[i_1,i_4] - \lambda[i_2,i_3]| \leq 1$ and
                $|\lambda[i_1,i_3] + \lambda[i_2,i_4] - \lambda[i_1,i_4] - \lambda[i_2,i_3]| \geq 2$ then
24:             $A :=$ implausible
25:         end if
26:     end for
27:   end for
28: end if;
29: Output $A$.
```

In Algorithm 10, we test if the paths between leaf vertices v_h, v_i and v_a, v_b, $h < i$ and $a < b$ are disjoint or intersecting in the current partial tree. For this at line 1, we compute the least common ancestor of v_h and v_i by Algorithm 11 and mark the vertices v_j on the path between v_a and v_b with an integer $z \in \{a, b\}$ by using Algorithm 7. Then from lines 3-9, we start from the leaf $v_x := v_h$ and set $v_x := p[v_x]$ if the mark of v_x is neither a nor b and in the other case, i.e., when the mark of v_x is a or b then output that the paths intersect. We repeat this process of setting $v_x := p[v_x]$ until v_x is the least common ancestor of v_h and v_i and the paths are disjoint. Similarly, from lines 10-17, we repeat the same process starting from $v_x := v_i$.

Algorithm 10 Testing Disjointness of Two Paths $\textsc{Disjoint}(h, i, a, b)$

Input: /* Global information: $G = (V(G) = \{v_1, v_2, \ldots, v_n\}, E(G)) \in \mathcal{G}_n$, the current tree $p[] = T_k \in \mathcal{T}(\{v_1, \ldots, v_k\})$, visit[] */
Four vertices $v_h, v_i, v_a, v_b \in \{v_1, v_2, \ldots, v_k\} \subseteq V(G)$ with $h < i$ and $a < b$.

Output: `intersect` if the paths between v_h, v_i and v_a, v_b in the current tree T_k intersect; `disjoint` otherwise.

```
1: $\ell$ :=LCA$(h, i)$; $\textsc{LcaTrace}(a, b)$;
2: $A :=$ disjoint; $x := h$;
3: while $x \neq \ell$ and $A =$ disjoint do
4:     if visit$[x] = a$ or visit$[x] = b$ then
5:         $A :=$ intersect
6:     else
7:         $x := p[x]$
8:     end if
9: end while;
10: $x := i$;
11: while $x \neq \ell$ and $A =$ disjoint do
12:     if visit$[x] = a$ or visit$[x] = b$ then
13:         $A :=$ intersect
14:     else
15:         $x := p[x]$
16:     end if
17: end while;
18: Output $A$.
```

In Algorithm 11, we compute the least common ancestor of two leaf vertices v_i and v_j with $i < j$ in the current tree if it has not been computed yet. From lines 2-5, we start with the vertex v_i by setting $v_x := v_i$ and mark all vertices on the path between v_i and v_1 except v_1 by the integer i. Then from lines 6-13, we start with the vertex v_j by setting $v_x := v_j$ and advance by $v_x := p[v_x]$ if v_x is not marked with i. We find the first vertex v_x on the path between v_j and v_1 with mark i starting from v_j and return v_x as the least common ancestor of v_i and v_j.

Algorithm 11 LCA(i, j): Finding Least Common Ancestor

Input: /* Global information: the current tree $p[] = T_k \in \mathcal{T}(\{v_1, \ldots, v_k\})$, the current partial coloring lca[], visit[] */
Two vertices $v_i, v_j \in \{v_1, v_2, \ldots, v_k\} \subseteq V(G)$, $i < j$.

Output: The least common ancestor lca$[i, j]$ of v_i and v_j in the current rooted tree $T_k \in \mathcal{T}(\{v_1, v_2, \ldots, v_k\})$.

```
 1: if lca$[i,j] = 0$ then
 2:     $x := i$;
 3:     while $x \neq 1$ do
 4:         visit$[x] := i$; $x := p[x]$
 5:     end while;
 6:     $x := j$;
 7:     while lca$[i,j] = 0$ do
 8:         if visit$[x] = i$ then
 9:             lca$[i,j] := x$
10:         else
11:             $x := p[x]$
12:         end if
13:     end while
14: end if;
15: Output lca$[i,j]$.
```

In Algorithm 12, for two leaves v_i and v_j with $i < j$ in the currrent tree we mark the vertices on the path between v_i and v_j with an integer $z \in \{i, j\}$ by computing the least common ancestor of v_i and v_j. From lines 1-4, we start with the vertex v_i by setting $v_x := v_i$ and mark all vertices on the path between v_i and v_1 except v_1 by the integer i. From lines 5-17, we start with the vertex v_j by setting $v_x := v_j$ and mark the vertex v_x by j and set $v_x := p[v_x]$ if v_x is not marked by i. Once we reach at the first vertex v_x that has been marked i, then we output v_x as the least common ancestor of v_i and v_j and mark all vertices on the path between $p[v_x]$ and v_1 by zero.

Algorithm 12 LCATRACE(i, j): Finding Least Common Ancestor

Input: /* Global information: $G = (V(G) = \{v_1, v_2, \ldots, v_n\}, E(G)) \in \mathcal{G}_n$, the current tree $p[] = T_k \in \mathcal{T}(\{v_1, \ldots, v_k\})$, visit[] */
Two vertices $v_i, v_j \in \{v_1, v_2, \ldots, v_k\} \subseteq V(G)$, $i < j$.

Output: The least common ancestor lca$[i, j]$ after marking the vertices v_x between v_i and v_j with visit$[x] \in \{i, j\}$ in the current rooted tree $T_k \in \mathcal{T}(\{v_1, v_2, \ldots, v_k\})$.

```
 1: $x := i$;
 2: while $x \neq 1$ do
 3:     visit$[x] := i$; $x := p[x]$
 4: end while;
 5: flag := 1; $x := j$;
 6: while $x \neq 1$ do
 7:     if visit$[x] \neq i$ then
 8:         visit[$x$] := $j$
 9:     **else**
10:         **if** visit[$x$] = $i$ and flag = 1 **then**
11:             lca[$i, j$] := $x$; flag := 0
12:         **else**
13:             visit[$x$] := 0
14:         **end if**
15:     **end if**;
16:     $x := p[x]$
17: **end while**;
18: Output lca[$i, j$].
```

5.3.1 Experimental Results for Nine Vertices

By incorporating the algorithm for generating feasible pairs in phase (II), instead of using phases (II-1) and (II-2), we generated PCGs with nine vertices. For our experiments, we used a PC with specifications Intel(R) Xeon(R) E5-1600v3 processor running at 3.00GHz, 64GB of memory, and Windows 7. We used the `IBM ILOG CPLEX 12.8` solver and find integer solutions for linear programs LP and DLP to avoid numerical errors generated by the solver while solving these formulations.

From [2], we obtained 261,080 connected graphs for $n = 9$ each of which are non-isomorphic. There are six binary trees in $\mathcal{T}_9$ that are easy to generate. The edge sets of each of these six trees T_1, T_2, T_3, T_4, T_5 and T_6 are listed in Table 5.3 with $V(T_i) = \{1, 2, \ldots, 16\}, i \in [1, 6]$ and representing an edge jk as j-k.

For an easy comparison of the PCG generated by the PCG generator, we used canonical representation of graphs obtained by `NAUTY` [47]. Recall that by [23] and [65] a graph is not an MNPCG if it (a) is not biconnected; (b) has two non-adjacent vertices that have the same neighbors; or (c) is a supergraph of some known MNPCG. Therefore we first removed all such graphs from the 261,080 graphs before phase (I-2).

We generated PCGs with our PCG generator in phase (I-2) for 10 days. As a result 7,108 graphs were left for which we used phase (I-3). We performed phase (I-3) in two steps. In the first step we fixed $\alpha = 300$ and time out to 60 secs. for the CPLEX solver. Thus in 18 days, we were able to confirm 4,603 graphs to be PCGs. In the second step, we fixed $\alpha = 300$ and time out to 300 secs. for the solver and confirmed another 583 graphs to be PCGs in 14 days. Therefore, 1,922 graphs were left after phase (1) for which we apply phase (II) to get definite proofs.

Table 5.3. Edge sets of the six trees in $\mathcal{T}_9$ with vertex set $\{1, 2, \ldots, 16\}$

i	Edge set of the tree T_i
1	1-10, 2-10, 3-11, 4-11, 5-12, 6-13, 7-14, 8-15, 9-16, 10-12, 11-13, 12-14, 13-15, 14-16, 15-16
2	1-10, 2-10, 3-13, 4-11, 5-11, 6-12, 7-12, 8-14, 9-16, 10-13, 11-14, 12-15, 13-14, 14-16, 15-16
3	1-10, 2-10, 3-11, 4-11, 5-12, 6-12, 7-13, 8-13, 9-16, 10-14, 11-14, 12-15, 13-15, 14-16, 15-16
4	1-10, 2-10, 3-11, 4-12, 5-12, 6-13, 7-14, 8-15, 9-15, 10-11, 11-13, 12-14, 13-16, 14-16, 15-16
5	1-10, 2-10, 3-11, 4-11, 5-12, 6-12, 7-14, 8-15, 9-15, 10-13, 11-13, 12-14, 13-16, 14-16, 15-16
6	1-10, 2-10, 3-11, 4-11, 5-12, 6-12, 7-13, 8-14, 9-15, 10-13, 11-13, 12-15, 13-16, 14-16, 15-16

Table 5.4. Summary of experimental results for graphs with nine vertices

Steps	# input graphs	# graphs with no decision	# MNPCGs	Time
Preprocessing and using PCG generator	261,080	7,108	0	10 days
Solving ILP$_{\text{suff}}$ for 60 secs.	7,108	2,505	0	18 days
Solving ILP$_{\text{suff}}$ for 300 secs.	2,505	1,922	0	14 days
Getting definite proof	1,922	0	1,494	46 days

The computation of phase (II) completed in 46 days and this phase confirmed 428 graphs to be PCGs. Therefore as a conclusion, we found that there are 1,494 graphs with $n = 9$ that are MNPCGs in 88 days. We show fifteen of these MNPCGs in Fig. 5.6. A summary of these experiments is given in Table 5.4. We summarize our results in Theorem 5.8.

Theorem 5.8. *For nine vertices, there are exactly 4, 35, 152, 289, 371, 337, 192, 85, 23, 5 and 1 MNPCGs with* $16, 17, \ldots, 25$, *and* 27 *edges, respectively, and no MNPCG with* β *edges* $\beta \notin [16, 25] \cup \{27\}$.

As a computational proof of Theorem 5.8, we provide the solution of the formulation DLP(G, T, I, λ) for each MIC4-free configuration (G, T, I, λ) at `https://www-or.amp.i.kyoto-u.ac.jp/~azam/MNPCG_9`.

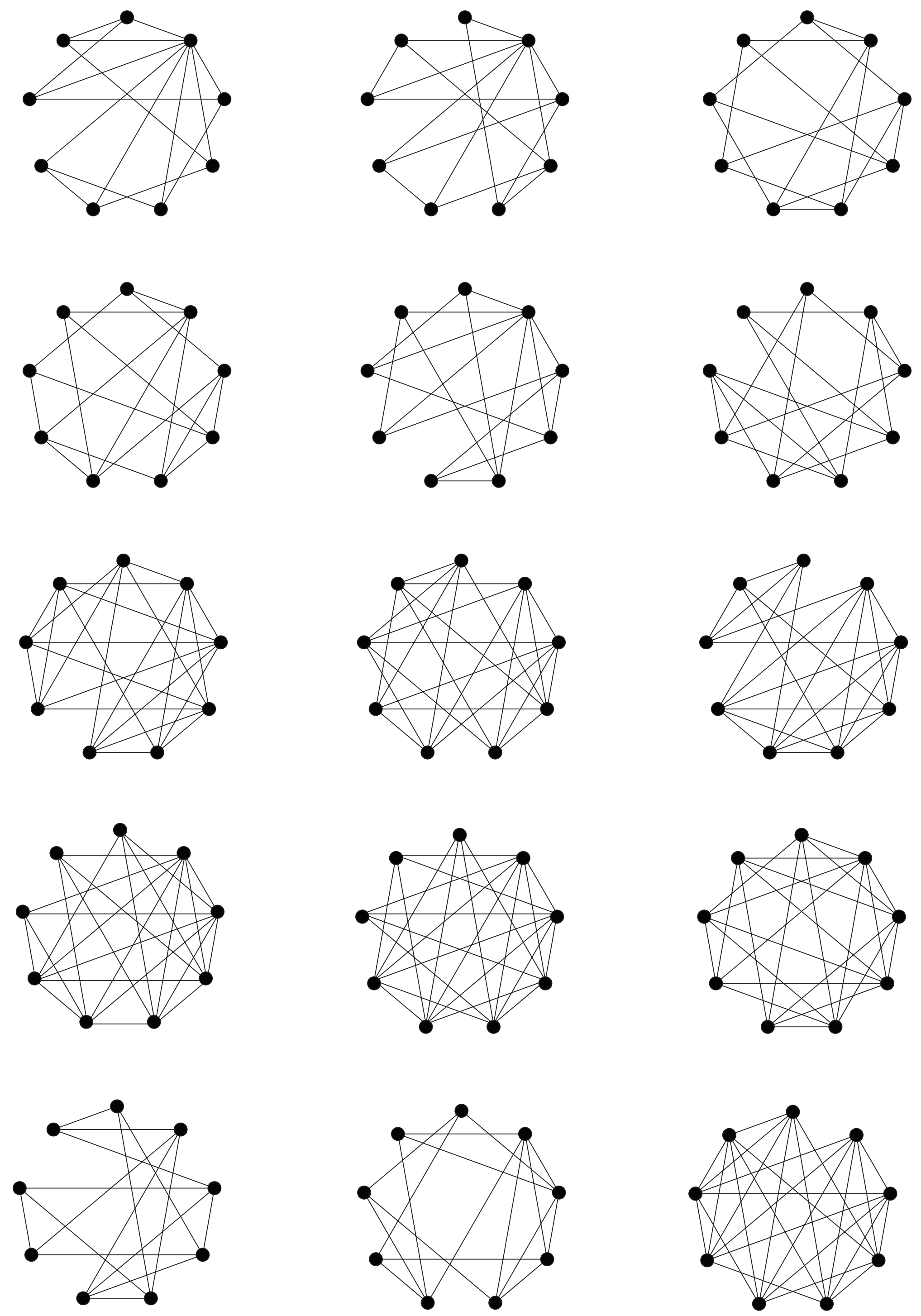


Figure 5.6. Fifteen MNPCGs with nine vertices discovered by our proposed method.

5.4 Concluding Remarks

We proposed a 2-phase method to enumerate PCGs with a given number of vertices. In phase (I), we dealt with the difficulty of a large number of configurations by developing a PCG generator using tree automorphisms and then constructing plausible configurations to prove a graph to be a PCG. In phase (II), we designed a branch-and-bound algorithm to enumerate all MIC4-free configurations and used LP formulations to handle the difficulty of infinite search space of weights and the construction of finite evidence. By using this method we proved that there are exactly seven and 1,494 MNPCGs with eight and nine vertices, respectively.

The wheel graph with $n \geq 9$ vertices is one of the sparsest graphs that is known to be an MNPCG [10] and is verified by our algorithm. It is interesting to investigate the minimum and maximum number of edges in an MNPCG with n vertices.

The illustrations of the MNPCGs in Fig. 5.6 that are discovered through this work, as well as those shown by Azam et al. [3] with eight vertices, Durocher et al. [27] and Baiocchi et al. [10] indicate that MNPCGs tend to be symmetric. Therefore it is natural to discover a relationship between the symmetry and membership of a graph in the class of PCG graphs.

By [2], there are 11,716,571 connected graphs with 10 vertices. Thus to generate all MNPCGs with 10 vertices efficiently, it would be interesting to further improve our method.

6 Conclusion

We focused on the enumeration of tree-like graphs with a given cycle rank, self-loops, and no multi-edges; and pairwise compatibility graphs (PCGs) with a given number of vertices.

Instead of traditional algorithm paradigms, we proposed a dynamic programming based method to count all tree-like graphs. We characterized the tree-like graphs in terms of canonical representation, which is then used to propose a method to generate all such graphs. Experimental results reveal that the proposed methods can count and generate the required graphs efficiently. As an application to our methods, we proved bounds on the number of tree-like polymer topologies and generated such topologies.

We proposed LP formulations to confirm if a graph is a PCG or not by providing finite-sized evidence. Furthermore, we characterized the configurations associated with a graph in terms of linear inequalities that gives us a necessary condition for a graph to be PCG with the given configuration. Based on these mathematical properties, we proposed a method to enumerate all PCGs with a given number of vertices. By using this method, we proved that there are exactly seven and 1,494 minimal non-PCGs with eight and nine vertices, respectively.

We hope that the results obtained in this study will be useful in future research and development in combinatorial mathematics, computational chemistry, bioinformatics, and other related fields.

Bibliography

[1] `http://sunflower.kuicr.kyoto-u.ac.jp/tools/enumol2/` [visited on June, 2019].

[2] `http://users.cecs.anu.edu.au/~bdm/data/graphs.html` [visited on June, 2019].

[3] N. A. Azam, M. Ito, A. Shurbevski, and H. Nagamochi. Enumerating all pairwise compatibility graphs with a given number of vertices based on linear programming. In *Proceedings of 2nd International Workshop on Enumeration Problems and Applications (WEPA)*, pages 5–8, Italy, WEPA2018 6c, November 2018. Pisa.

[4] N. A. Azam, R. Chiewvanichakorn, F. Zhang, A. Shurbevski, H. Nagamochi, and T. Akutsu. A method for the inverse QSAR/QSPR based on artificial neural networks and mixed integer linear programming. In *Proceedings of the 13th International Joint Conference on Biomedical Engineering Systems and Technologies – Volume 3: BIOINFORMATICS*, 2020.

[5] N. A. Azam, A. Shurbevski, and H. Nagamochi. Enumerating tree-like graphs and polymer topologies with a given cycle rank. *Entropy*, 22(11): 1295, 2020.

[6] N. A. Azam, A. Shurbevski, and H. Nagamochi. A method for enumerating pairwise compatibility graphs with a given number of vertices. *Discrete Applied Mathematics*, 2020.

[7] N. A. Azam, A. Shurbevski, and H. Nagamochi. On the enumeration of minimal non-pairwise compatibility graphs. In *International Computing and Combinatorics Conference*, pages 372–383. Springer, 2020.

[8] N. A. Azam, A. Shurbevski, and H. Nagamochi. An efficient algorithm to count tree-like graphs with a given number of vertices and self-loops. *Entropy*, 22(9):923, 2020.

[9] N. A. Azam, J. Zhu, R. Ido, H. Nagamochi, and T. Akutsu. Experimental results of a dynamic programming algorithm for generating chemical isomers based on frequency vectors. In *Fourth International Workshop on Enumeration Problems and Applications (WEPA)*, pages 7–10, Israel, WEPA2020 15, December 2020. Israel.

[10] P. Baiocchi, T. Calamoneri, A. Monti, and R. Petreschi. *Some classes of graphs that are not PCGs.* Theoretical Computer Science, 2019. URL `https://doi.org/10.1016/j.tcs.2019.05.017`.

[11] M. Beaudouin-Lafon, S. Chen, N. Karst, D. S. Troxell, and X. Zheng. Pairwise compatibility graphs: complete characterization for wheels. *Involve, a Journal of Mathematics*, 12(5):871–882, 2019.

[12] T. Beyer and S. M. Hedetniemi. Constant time generation of rooted trees. *SIAM Journal on Computing*, 9(4):706–712, 1980.

[13] L. C. Blum and J.-L. Reymond. 970 million druglike small molecules for virtual screening in the chemical universe database gdb-13. *Journal of the American Chemical Society*, 131(25):8732–8733, 2009.

[14] B. G. Buchanan and E. A. Feigenbaum. Dendral and meta-dendral: Their applications dimension. In *Readings in artificial intelligence*, pages 313–322. Elsevier, 1981.

[15] L. Bytautas and D. J. Klein. Chemical combinatorics for alkane-isomer enumeration and more. *Journal of chemical information and computer sciences*, 38(6):1063–1078, 1998.

[16] T. Calamoneri and R. Petreschi. On pairwise compatibility graphs having dilworth number two. *Theoretical Computer Science*, 524:34–40, 2014.

[17] T. Calamoneri and R. Petreschi. On pairwise compatibility graphs having dilworth number k. *Theoretical Computer Science*, 547:82–89, 2014.

[18] T. Calamoneri and B. Sinaimeri. Pairwise compatibility graphs: A survey. *SIAM Review*, 58(3):445–460, 2016.

[19] T. Calamoneri, R. Petreschi, and B. Sinaimeri. On relaxing the constraints in pairwise compatibility graphs. In *International Workshop on Algorithms and Computation*, pages 124–135. Springer, 2012.

[20] T. Calamoneri, D. Frascaria, and B. Sinaimeri. All graphs with at most seven vertices are pairwise compatibility graphs. *The Computer Journal*, 56 (7):882–886, 2013.

[21] T. Calamoneri, R. Petreschi, and B. Sinaimeri. On the pairwise compatibility property of some superclasses of threshold graphs. *Discrete Mathematics, Algorithms and Applications*, 5(02):1360002, 2013.

[22] T. Calamoneri, E. M. T., R. Petreschi, and B. Sinaimeri. Exploring pairwise compatibility graphs. *Theoretical Computer Science*, 468:23–26, 2013.

[23] T. Calamoneri, A. Frangioni, and B. Sinaimeri. Pairwise compatibility graphs of caterpillars. *The Computer Journal*, 57:1616–1623, 2014.

[24] A. Cayley. On the analytical forms called trees, with application to the theory of chemical combinations. *Rep. Brit. Assoc. Advance. Sci*, 45:257–305, 1875.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT press, 2009.

[26] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1036–1050, 2005.

[27] S. Durocher, D. Mondal, and M. S. Rahman. On graphs that are not pcgs. *Theoretical Computer Science*, 571:78–87, 2015.

[28] J.-L. Faulon, C. J. Churchwell, and D. P. Visco. The signature molecular descriptor. 2. enumerating molecules from their extended valence sequences. *Journal of Chemical Information and Computer Sciences*, 43(3):721–734, 2003.

[29] T. Fink and J.-L. Reymond. Virtual exploration of the chemical universe up to 11 atoms of c, n, o, f: assembly of 26.4 million structures (110.9 million stereoisomers) and analysis for new ring systems, stereochemistry, physicochemical properties, compound classes, and drug discovery. *Journal of chemical information and modeling*, 47(2):342–353, 2007.

[30] H. Fujiwara, J. Wang, L. Zhao, H. Nagamochi, and T. Akutsu. Enumerating treelike chemical graphs with given path frequency. *Journal of Chemical Information and Modeling*, 48(7):1345–1357, 2008.

[31] K. Funatsu and S.-i. Sasaki. Recent advances in the automated structure elucidation system, chemics. utilization of two-dimensional nmr spectral information and development of peripheral functions for examination of candidates. *Journal of Chemical Information and Computer Sciences*, 36(2): 190–204, 1996.

[32] D. Gale. The theory of linear economic models. *MacGraw-Hill*, 1960.

[33] H. Galina and M. M. Sysło. Some applications of graph theory to the study of polymer configuration. *Discrete applied mathematics*, 19(1-3):167–176, 1988.

[34] R. Gugisch, A. Kerber, A. Kohnert, R. Laue, M. Meringer, C. Rücker, and A. Wassermann. Molgen 5.0, a molecular structure generator. In *Advances in mathematical chemistry and applications*, pages 113–138. Elsevier, 2015.

[35] L. H. Hall, R. S. Dailey, and L. B. Kier. Design of molecules from quantitative structure-activity relationship models. 3. role of higher order path counts: Path 3. *Journal of Chemical Information and Computer Sciences*, 33(4): 598–603, 1993.

[36] T. Haruna, T. Horiyama, and K. Shimokawa. On the enumeration of polymer topologies. *IPSJ SIG Technical Report*, 2017-Al-162(5), 2017.

[37] Y. Ishida, L. Zhao, H. Nagamochi, and T. Akutsu. Improved algorithms for enumerating tree-like chemical graphs with given path frequency. *Genome Informatics*, 21:53–64, 2008.

[38] Y. Ishida, Y. Kato, L. Zhao, H. Nagamochi, and T. Akutsu. Branch-and-bound algorithms for enumerating treelike chemical graphs with given path frequency using detachment-cut. *Journal of chemical information and modeling*, 50(5):934–946, 2010.

[39] R. Ito, N. A. Azam, C. Wang, A. Shurbevski, H. Nagamochi, and T. Akutsu. A novel method for the inverse qsar/qspr to monocyclic chemical compounds based on artificial neural networks and integer programming. In *Advances in Computer Vision and Computational Biology.* Springer Nature - Research Book Series [To appear], 2020.

[40] W. Jin, R. Barzilay, and T. Jaakkola. Junction tree variational autoencoder for molecular graph generation. In *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[41] C. Jordan. Sur les assemblages de lignes. *J. Reine Angew. Math*, 70(185): 81, 1869.

[42] P. E. Kearney, J. I. Munro, and D. Phillips. Efficient generation of uniform samples from phylogenetic trees. *International Workshop on Algorithms in Bioinformatics*, pages 177–189, 2003.

[43] L. B. Kier, L. H. Hall, and J. W. Frazer. Design of molecules from quantitative structure-activity relationship models. 1. information transfer between path and vertex degree counts. *Journal of Chemical Information and Computer Sciences*, 33(1):143–147, 1993.

[44] D. E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions.* Addison-Wesley Professional, 2005.

[45] J. Lim, S.-Y. Hwang, S. Moon, S. Kim, and W. Y. Kim. Scaffold-based molecular design with a graph generative model. *Chemical Science*, 11(4): 1153–1164, 2020.

[46] R. Masui, A. Shurbevski, and H. Nagamochi. Enumeration of unlabeled tree by dynamic programming. *Technical Report No. 2019-003, Department of Applied Mathematics and Physics, Kyoto University, http://www.amp.i.kyoto-u.ac.jp/tecrep/ps-file/2019/2019-003.pdf*, 2019.

[47] B. D. McKay and A. Piperno. Practical graph isomorphism, ii. *Journal of Symbolic Computation*, 60:94–112, 2014.

[48] S. Mehnaz and M. S. Rahman. Pairwise compatibility graphs revisited. In *2013 International conference on informatics, electronics and vision (ICIEV)*, pages 1–6. IEEE, 2013.

[49] O. Méndez-Lucio, B. Baillif, D.-A. Clevert, D. Rouquié, and J. Wichard. De novo generation of hit-like molecules from gene expression signatures using artificial intelligence. *Nature communications*, 11(1):1–10, 2020.

[50] M. Meringer and E. L. Schymanski. Small molecule identification with molgen and mass spectrometry. *Metabolites*, 3(2):440–462, 2013.

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

[52] S.-i. Nakano and T. Uno. Efficient generation of rooted trees. *National Institute for Informatics (Japan), Tech. Rep. NII-2003-005E*, 8, 2003.

[53] S.-i. Nakano and T. Uno. A simple constant time enumeration algorithm for free trees. *PSJ SIGNotes ALgorithms*, (091-002), 2003.

[54] S.-i. Nakano and T. Uno. Generating colored trees. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 249–260. Springer, 2005.

[55] J. E. Peironcely, M. Rojas-Chertó, D. Fichera, T. Reijmers, L. Coulier, J.-L. Faulon, and T. Hankemeier. Omg: open molecule generator. *Journal of cheminformatics*, 4(1):21, 2012.

[56] G. Pólya. Kombinatorische anzahlbestimmungen für gruppen, graphen und chemische verbindungen. *Acta mathematica*, 68(1):145–254, 1937.

[57] G. Polya and R. C. Read. *Combinatorial enumeration of groups, graphs, and chemical compounds*. Springer Science & Business Media, 2012.

[58] S. A. Salma and M. S. Rahman. Triangle-free outerplanar 3-graphs are pairwise compatibility graphs. In *International Workshop on Algorithms and Computation*, pages 112–123. Springer, 2012.

[59] M. Shimizu, H. Nagamochi, and T. Akutsu. Enumerating tree-like chemical graphs with given upper and lower bounds on path frequencies. In *BMC bioinformatics*, volume 12, page S3. Springer, 2011.

[60] M. Suzuki, H. Nagamochi, and T. Akutsu. A 2-phase algorithm for enumerating tree-like chemical graphs satisfying given upper and lower bounds. *IPSJ SIG Tech. Reports*, 28(17):1–8, 2012.

[61] M. Suzuki, H. Nagamochi, and T. Akutsu. Efficient enumeration of monocyclic chemical graphs with given path frequencies. *Journal of Cheminformatics*, 6(1):31, 2014.

[62] Y. Tezuka and H. Oike. Topological polymer chemistry. *Progress in polymer science*, 27(6):1069–1122, 2002.

[63] M. Vogt and J. Bajorath. Chemoinformatics: a view of the field and current trends in method development. *Bioorganic & medicinal chemistry*, 20(18): 5317–5323, 2012.

[64] R. A. Wright, B. Richmond, A. Odlyzko, and B. D. McKay. Constant time generation of free trees. *SIAM Journal on Computing*, 15(2):540–548, 1986.

[65] M. Xiao and H. Nagamochi. Some reduction operations to pairwise compatibility graphs. *Information Processing Letters*, 153:105875, 2020.

[66] M. N. Yanhaona, K. T. Hossain, and M. S. Rahman. Pairwise compatibility graphs. *Journal of Applied Mathematics and Computing*, 30(1-2):479–503, 2009.

[67] N. M. Yanhaona, M. S. Bayzid, and M. S. Rahman. Discovering pairwise compatibility graphs, discrete mathematics. *Algorithms and Applications*, 2 (4):479–503, 2010.

[68] F. Zhang, J. Zhu, R. Chiewvanichakorn, A. Shurbevski, H. Nagamochi, and T. Akutsu. A new integer linear programming formulation to the inverse qsar/qspr for acyclic chemical compounds using skeleton trees. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 433–444. Springer, 2020.

[69] J. Zhu, C. Wang, A. Shurbevski, H. Nagamochi, and T. Akutsu. A novel method for inference of chemical compounds of cycle index two with desired properties based on artificial neural networks and integer programming. *Algorithms*, 13(5):124, 2020.

[70] B. Zhuang and H. Nagamochi. Constant time generation of trees with degree bounds. In *9th International Symposium on Operations Research and Its Applications*, pages 183–194, 2010.

[71] B. H. Zimm and W. H. Stockmayer. The dimensions of chain molecules containing branches and rings. *The Journal of Chemical Physics*, 17(12): 1301–1314, 1949.

List of the Author's Work

Publications Related to the Dissertation

[1] N. A. Azam, A. Shurbevski, and H. Nagamochi. Counting tree-like graphs with a given number of vertices and self-loops. In *3rd International Workshop on Enumeration Problems and Applications (WEPA)*, Japan, 28-31 October 2019, Awaji Island.

[2] N. A. Azam, A. Shurbevski, and H. Nagamochi. An efficient algorithm to count tree-like graphs with a given number of vertices and self-loops. *Entropy*, 22(9):923, 2020.

[3] N. A. Azam, A. Shurbevski, and H. Nagamochi. Enumerating tree-like graphs and polymer topologies with a given cycle rank. *Entropy*, 22(11): 1295, 2020.

[4] N. A. Azam, M. Ito, A. Shurbevski, and H. Nagamochi. Enumerating all pairwise compatibility graphs with a given number of vertices based on linear programming. In *2nd International Workshop on Enumeration Problems and Applications (WEPA)*, pages 5–8, Italy, WEPA2018 6c, November 2018. Pisa.

[5] N. A. Azam, A. Shurbevski, and H. Nagamochi. A method for enumerating pairwise compatibility graphs with a given number of vertices. *Discrete Applied Mathematics*, 2020
Copyright © 2020 Elsevier.

[6] N. A. Azam, A. Shurbevski, H. Nagamochi. An improved method for enumerating pairwise compatibility graphs with a given number of vertices. In *Asian Association for Algorithms and Computation (AAAC)*, South Korea, 18–22 April 2019, Seoul.

[7] N. A. Azam, A. Shurbevski, and H. Nagamochi. On the enumeration of minimal non-pairwise compatibility graphs. In *International Computing and Combinatorics Conference*, pages 372–383. Springer, 2020
Copyright © 2020 Springer Nature Switzerland.

[8] N. A. Azam, A. Shurbevski, and H. Nagamochi. On the enumeration of minimal non-pairwise compatibility graphs. *Journal of Combinatorial Optimization* [submitted].