Generalization of the Consecutive-ones Property

A THESIS

submitted by

ANJU SRINIVASAN

for the award of the degree of

MASTER OF SCIENCE *by Research*

from the department of

COMPUTER SCIENCE AND ENGINEERING

at

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Guindy, Chennai - 600036

FEBRUARY 2012

THESIS CERTIFICATE

This is to certify that the thesis titled **Generalization of the Consecutive-ones Property**, submitted by **Anju Srinivasan**, to the **Indian Institute of Technology Madras**, for the award of the degree of **Master of Science *by Research***, is a bona fide record of the research work done by her under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. N. S. Narayanaswamy
Research Guide
Associate Professor
Dept. of Computer Science & Engineering
IIT Madras – 600 036

Chennai
March 23, 2013

ACKNOWLEDGEMENTS

Like any body of work, big or small, this thesis and the study behind it could not have been done by me without the support of many people and things professionally and personally. I believe everyone and everything I encountered in the last two years and seven months has directly or indirectly helped me conclude this work. Following are the most obvious and direct help that I explicitly acknowledge.

I have deep gratitude for my research advisor Dr. N. S. Narayanaswamy. I have been very fortunate to be associated with a teacher with such remarkable patience and tenacity with his students that brings out the possible best in them. He let us work with the right kind of liberty that is needed to discover and practice one's own method of research and discipline. I would like to thank him much for all his brilliant ideas, many discussions (at times repeated and with no judgement) and perpetual words of encouragement. I doubt if I would have learned my research without his persevering guidance. I would like to thank all fellow students in my research group for frequent technical discussions and faithful reviews; also my friends outside AIDB lab, for all caffeine-induced discussions and de-stressing arguments.

Institute of Mathematical Sciences, for all the mind-bending lectures that set a foundation to my research. I would especially like to thank Prof. R. Ramanujam for motivating and encouraging my interest in interdisciplinary research.

My husband, Zabil, for his amazing support and encouragement to take a break from my career and pursue my higher education. Moreover, for putting up patiently with an approximate solution to our two-body problem in these years.

My parents, Srinivasan and Seena, for always being there and helping me in ways only parents can help. My sister, Renu, for her precocious optimism. Devi and Rinchen, for being a hotline for procrastination-induced panic attacks and reminding me that I can indeed complete my M. S. ! LMN – Lakshmi, Mahathi and Nari, for being the closest

thing to a family on campus always and especially whenever I defaulted to being a hermit (usually a famished one). Jai, J. K. , Shivani and Sourab for having an open door policy for me any day any time that I was in Chennai and incentivizing my visits with great fun and awesome food.

And last but certainly not the least, IITM and its wonderful campus for inspiring me to unlearn my old methods of learning and helping me discover and enjoy research.

[...] a mathematical experience was aesthetic in nature, an epiphany in Joyce's original sense. These moments appeared in proof-completions, or maybe algorithms. Or like a gorgeously simple solution you suddenly see after filling half a notebook with gnarly attempted solutions. It was really an experience of what I think Yeats called "the click of a well made box".

D. F. W.

Math Graffiti: Kilroy wasn't Haar. Free the group. Nuke the kernel. Power to the n.
$N = 1 \Rightarrow P = NP$

Concrete Mathematics (margin notes)

The process of preparing programs for the digital computer is especially attractive, not only because it can be economically and scientifically rewarding, but also because it can be an aesthetic experience much like composing poetry or music.

The Art of Computer Programming, D. E. K.

L'art c'est la solution au chaos.

(Art is the solution to chaos.)

myth·os $|'\text{mi}\ \theta\ \bar{o}\text{s}|$

a set of beliefs or assumptions about something : *the rhetoric and mythos of science create the comforting image of linear progression toward truth.*

New Oxford American Dictionary, 2nd Ed.

Anything that happens, happens. Anything that, in happening, causes something else to happen, causes something else to happen. Anything that, in happening, causes itself to happen again, happens again. It doesn't necessarily do it in chronological order, though.

The Salmon of Doubt, D. N. A.

Everything in its right place.

Kid A, Radiohead, T. Y.

ABSTRACT

Keywords: *consecutive ones property, algorithmic graph theory, hypergraph isomorphism, interval labeling*

Consecutive-ones property is a non-trivial combinatorial property of binary matrices that has been studied widely in the literature for over past 50 years. Detection of COP in a matrix is possible efficiently and there are several algorithms that achieve the same. This thesis documents the work done on an extension of COP. We extend COP from the equivalent interval assignment problem to what we call *Tree Path Labeling (TPL)* problem. Inspired by the idea that COP is akin to assigning intervals to sets in a set system, we ask the question what if the assignments to sets are made not of intervals but of paths from any tree? While we show that TPL is in NP, we obtain a more efficient algorithm for a special case tree class called *k-subdivided stars* as well as for characterizing a feasible TPL. For extension of COP to arbitrary trees, we analyse the structure of a hypergraph (set system / columns of matrix) using the idea of prime submatrices and yield an algorithm for a feasible path labeling if one indeed exists. These new results rigorously prove the natural extension (to trees) of the ICPIA characterization as well as makes connections to graph isomorphism, namely path graph isomorphism.

Moreover, this thesis discusses the significance of COP and the evolution of existing data structures and algorithms used in detection of COP over the years – from PQ-trees to Intersection Cardinality Preserving Interval Assignment (ICPIA) data structure. We draw conclusions and connections of which some of them (for instance, PQR-tree and generalized PQ-tree) have not be observed before, to the best of our knowledge. This includes the fact that COP can be tested in logspace.

CONTENTS

Abstract **vii**

Contents **ix**

List of Tables **xi**

List of Figures **xiii**

Abbreviations and Notations **xv**

1 Introduction **1**

1.1 Organization of the document 1

1.2 Illustration of the problem 2

1.2.1 Special case 3

1.3 Basic preliminaries 5

1.3.1 Matrices 5

1.3.2 Sets 6

1.3.3 Graphs 7

1.4 Brief Survey 8

1.4.1 Matrices with COP 8

1.4.2 Optimization problems in COP 10

1.5 Generalization of COP 11

1.6 Summary of new results 12

2 COP – A Survey **15**

2.1 COP in Graph Theory 16

2.2 Matrices with COP 18

2.2.1 Tucker's forbidden submatrices for COP 21

2.2.2 Booth and Lueker's PQ-tree – a linear COT algorithm 22

2.2.3 PQR-tree – COT for set systems 25

2.2.4 PC-tree– a generalization of PQ-tree 29

2.2.5 ICPIA - a set cardinality based COP test 31

2.3 Matrices without COP 33

2.3.1 Finding forbidden submatrices 33

2.3.2 Incompatibility graph - a certificate for no COP 35

3 Tree Path Labeling **37**

3.1 Summary of problems 37

3.2 Preliminaries 39

3.2.1 Set systems and Hypergraphs 40

3.2.2 Path Labeling and Path Hypergraphs 41

3.2.3 Overlap Graphs and Marginal Hyperedges 43

3.2.4 Miscellaneous 43

3.3 Characterization of FTPL 44

3.4 Solution to study group accommodation problem 52

3.5 Special target trees 54

3.5.1 Target tree is a Path 55

3.5.2 Target tree is a k-subdivided Star 55

3.6 TPL on arbitrary trees 60

3.7 Complexity 69

3.7.1 Time complexity of TPL 69

3.7.2 Consecutive Ones Testing is in Logspace 69

4 Conclusion **71**

Bibliography **75**

LIST OF TABLES

1.1 Students and study groups in *Wallace Studies Institute* 2

1.2 A solution to study group accommodation problem 2

2.1 Relationship between graph classes and graph matrices with COP or CROP. 19

2.2 A brief history of COP research 21

2.3 Comparison of theory of PQR-tree, gPQ-tree, generalized PQ-tree 30

LIST OF FIGURES

1.1 *Infinite Loop* street map. 4

1.2 *Infinite Loop* street map with study group routes allocated. 4

1.3 Solution to the student accommodation problem. 4

1.4 Matrices with and without COP. 6

1.5 Examples of k-subdivided stars. (a) $k = 0$ (b) $k = 2$ 13

2.1 Matrices defined in Def. 2.1.1 17

2.2 Tucker's forbidden subgraphs 22

2.3 Tucker's forbidden submatrices 23

2.4 An example for PQ-tree 24

2.5 PC-tree 31

3.1 Hypergraphs and set systems 41

3.2 Problem solution part 1 52

3.3 Problem solution part 2 53

3.4 Problem solution part 3 53

3.5 Problem solution part 4 54

3.6 Problem solution part 5 54

3.7 Problem solution part 6 54

3.8 Problem solution part 7 55

3.9 (a) 8-subdivided star with 7 rays (b) 3-subdivided star with 3 rays 56

3.10 Partial order on prime submatrices - containment partition 67

3.11 Partial order on prime submatrices - target tree 67

ABBREVIATIONS AND NOTATIONS

COP	Consecutive-ones Property
COT	Consecutive-ones property Testing
CROP	CiRcular-ones Property
e. g.	*exempli gratia*
i. e.	*id est*
ICPIA	Intersection Cardinality Preservation Interval Assignment
ICPPL	Intersection Cardinality Preserved Path Labeling
QED	*quod erat demonstrandum*

2^U	Powerset of set U
$[n]$	The set $\{1, 2, \ldots, n\}$ for any positive integer n
$\emptyset$	Empty set

CHAPTER 1

INTRODUCTION

Consecutive-ones property is a non-trivial property of binary matrices that has been studied widely in the literature for over past 50 years. Detection of COP in a matrix is possible efficiently and there are several algorithms that achieve the same. This thesis documents the work done on an extension of COP extended from the equivalent interval assignment problem in [NS09]. These new results rigorously prove a natural extension (to trees) of their characterization as well as makes connections to graph isomorphism, namely path graph isomorphism.

1.1 Organization of the document

Chapter 1 introduces the area of research and the problems addressed in this thesis. Chapter 2 gives a more detailed survey briefed in Section 1.4. Chapter 3 details all the results obtained to the problems of this thesis and finally the conclusion of the thesis is discussed in Chapter 4.

In this chapter, Section 1.2 introduces the main problem of this thesis by way of an illustration. Section 1.3 lays out a few general definitions that are helpful in understanding the rest of the chapter. Section 1.4 gives a brief survey of COP and optimization problems related to it followed by motivation for the thesis in Section 1.5. Section 1.6 presents a summary of our results on the extension of COP namely, the tree path labeling problem.

U	=	{**Pa**, **Pi**, **Sn**, **Wo**, **Vi**, **Li**, **Ch**, **Sa**, **Fr**, **Sc**, **Lu**}
$\mathcal{F}$	=	$\{\mathbb{B}, \mathbb{T}, \mathbb{W}, \mathbb{F}\}$
$\mathbb{B}$	=	{**Ch**, **Sa**, **Fr**, **Sc**, **Lu**}
$\mathbb{T}$	=	{**Pa**, **Pi**, **Vi**, **Ch**}
$\mathbb{W}$	=	{**Sn**, **Pi**, **Wo**}
$\mathbb{F}$	=	{**Vi**, **Li**, **Ch**, **Fr**}
n	=	$\lvert U\rvert = 11$
m	=	$\lvert\mathcal{F}\rvert = 4$

Table 1.1: Students and study groups in *Wallace Studies Institute*

1.2 Illustration of the problem

A group of students, **Pa***tricia*, **Pi***gpen*, **Sn***oopy*, **Wo***odstock*, **Vi***olet*, **Li***nus*, **Ch***arlie*, **Sa***lly*, **Fr***anklin*, **Sc***hröeder* and **Lu***cy* enroll at the *Wallace Studies Institute* for a liberal arts programme. As part of their semester thesis, they pick a body of work to study and form the namesake study groups, *"*$\mathbb{B}$*rief Interviews with Hideous Men"* [Wal99], *"The String* $\mathbb{T}$*heory"* [Wal96], *"[*$\mathbb{W}$*]Rhetoric and the Math Melodrama"* [Wal00] and *"*$\mathbb{F}$*ate, Time, and Language: An Essay on Free Will"* [Wal10]. A student will be in at least one study group and may be in more than one. For instance, as will be seen later, **Fr***anklin* studies both *"*$\mathbb{B}$*rief Interviews with Hideous Men"* and *"*$\mathbb{F}$*ate, Time, and Language: An Essay on Free Will"* while **Wo***odstock* studies only *"[*$\mathbb{W}$*]Rhetoric and the Math Melodrama"*.

Let U and $\mathcal{F}$ represent the set of students and the set of study groups respectively and the integers n and m denote the total number students and study groups respectively. In relation to this example, these are defined in Table 1.1. Also given there is the study group allocation to students.

The campus has a residential area *Infinite Loop* that has n single occupancy apartments reserved for the study groups' accommodation. All these apartments are located such that the streets connecting them do *not* form loops. Figure 1.1 shows the street map for *Infinite Loop*. It may be noted that as a graph, it classifies as a tree.

A natural question would be to find how the students should be allocated apartments such that each study group has the least distance to travel for a discussion? More

T	=	*Street map tree of* Infinite Loop
$V(T)$	=	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$
$\mathcal{P}$	=	$\{R\mathbb{B}, R\mathbb{T}, R\mathbb{W}, R\mathbb{F}\}$
$R\mathbb{B}$	=	$\{9, 1, 5, 3, 11\}$
$R\mathbb{T}$	=	$\{7, 2, 6, 5\}$
$R\mathbb{W}$	=	$\{8, 2, 4\}$
$R\mathbb{F}$	=	$\{10, 6, 5, 3\}$
n	=	$\lvert V\rvert = 11$
m	=	$\lvert\mathcal{P}\rvert = 4$
ℓ	=	*Study group to route mapping*
$\ell(\mathbb{X})$	=	$R\mathbb{X}$ for all $\mathbb{X} \in \mathcal{F}$

Apartment allocation (ϕ)	
1	**Sa**
2	**Pi**
3	**Fr**
4	**Wo**
5	**Ch**
6	**Vi**
7	**Pa**
8	**Sn**
9	**Lu**
10	**Li**
11	**Sc**

Table 1.2: A solution to study group accommodation problem

specifically, we are interested in the problem with additional conditions, namely, that all the students in a study group must be next to each other; in other words, for one student to reach another fellow study group member's apartment (for all study groups the student is part of), she must not have to pass the apartment of any student who is not in that study group. To further elucidate, the apartments of students of any study group must be arranged in an exclusive unfragmented path on the street map. Exclusivity here means that the path must not have apartments from other study groups (unless that apartment is also part of *this* study group).

An intuitive approach to this problem would be to first find the paths that each study group decides to inhabit and then refine the allocation to individual students. A feasible allocation of exclusive routes to study groups is illustrated in Figure 1.1. The students' allocation of apartments that obeys this route allocation is shown in Figure 1.3. Table 1.2 shows the same solution set theoretically. How this is algorithmically computed is the focus of this thesis.

1.2.1 Special case

As a special case of the study group accommodation problem, suppose all the apartments are on the same street or if they are all lined up on a single path, the street map becomes a tree that is just a path. Then the problem becomes what is called an *interval assignment problem*. The idea of interval assignment may not be obvious here; hence to see this, consider a different problem in *Wallace Studies Institute* where the classes for these study groups courses need to be scheduled during a day (or a week or any time period). Each study group has a bunch of courses associated with it some of which may be shared by two or more study groups. It is mandatory that a student who is a member of a study group takes all the courses associated with that group. There are slots during the day for classes to be held and the problem is to allocate class slots to courses such that all the classes of a study group are consecutive. The parallels between this class allocation problem and the accommodation problem can be seen as follows. The set U here, are the courses offered (say Course 101 *"Influence of post modernism in Wallace's work"*, Course 102 *"A study on fragmented prose method"* and so on). In this variation of the problem, the collection $\mathcal{F}$ is the set of study groups but the study groups are filled by course IDs (in place of students in the earlier example). For instance, Course 101 is mandatory for all study groups $\mathbb{B}$, $\mathbb{T}$, $\mathbb{W}$, $\mathbb{F}$ and Course 102 is mandatory for only the $\mathbb{B}$ group) and so on. The sequence of class slots for the day (or week or any time period) is analogous to the street map in the accommodation problem. It is quite obvious now why this version of the problem (where the "target graph" is a path and not any tree) is called an interval assignment problem.

The interval assignment problem to a set system is equivalent to the consecutive-

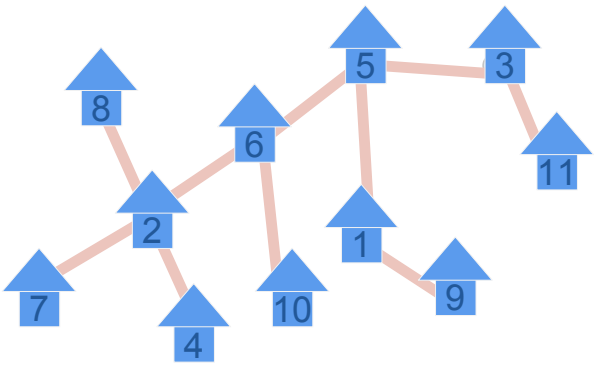


Figure 1.1: *Infinite Loop* street map.

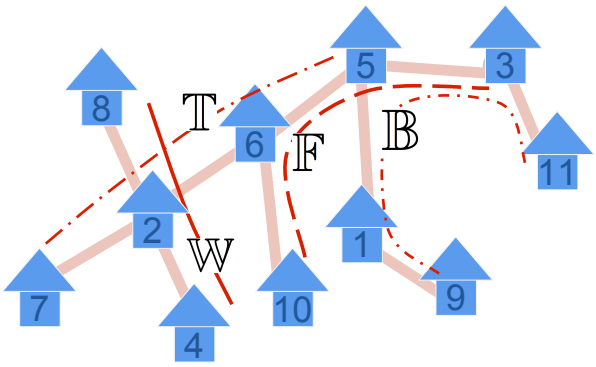


Figure 1.2: *Infinite Loop* street map with study group routes allocated.

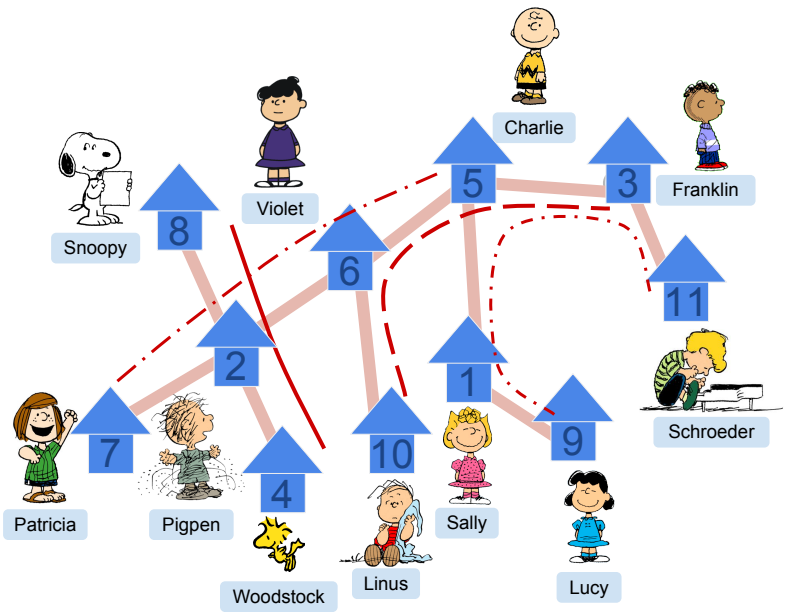


Figure 1.3: Individual allocation of apartments to students in *Infinite Loop* that meets the requirements stated before.

Peanuts images © Charles Schulz

ones property (COP) problem in binary matrices[Hsu02, NS09]. The COP problem is to rearrange rows (columns) of a binary matrix in such a way that every column (row) has its **1**s occur consecutively. If this is possible the matrix is said to have the COP. COP is a well researched combinatorial problem and has several positive results on tests for it and computing the COP permutation (i.e. the course schedule in the above illustration) which will be surveyed later in this document. Hence we are interested in extensions of COP, more specifically, the extension of interval assignment problem to tree path assignment problem (which is illustrated by the study group accommodation problem).

1.3 Basic preliminaries

Here we see some basic definitions and conventions necessary for the contents of this thesis.

1.3.1 Matrices

If n is a positive integer, $[n]$ denotes the set $\{1, 2, ...n\}$.

Definition 1.3.1 (Binary matrix). Let M be an $n \times m$ matrix. $m_{i,j}$ denotes its (i, j)th element, i.e. element at ith row and jth column. M is a **binary matrix** if each of its element is 0 or **1**; for all $i \in [n]$ and $j \in [m]$, $m_{i,j} \in \{0, \mathbf{1}\}$

Definition 1.3.2 (Permutation). A **permutation** λ of a set $X = \{x_1, x_2, \ldots x_n\}$ is a bijection $\lambda : X \to X$. λ may be written as a sequence $x_{i_1} x_{i_2} \ldots x_{i_n}$, where $i_j \in [n]$ to mean that $\lambda(x_j) = x_{i_j}$. This document uses both notations as convenient in context.

The idea of consecutive-ones property of binary matrices was mentioned a few times in earlier sections. Now we will see the formal definition of COP in Definition 1.3.3.

Definition 1.3.3 (Consecutive-ones property (on columns)). Let M be a binary matrix.

1. A **block of 1s (block of 0s)** in a column of M is a maximal set of consecutive **1**-entries (0-entries) in this column.
2. M has the **strong consecutive-ones property (strong COP)** if in every column the **1**s appear consecutively, i. e. if every column contains at most one block of **1**s.
3. M has the **consecutive-ones property** if its rows can be permuted in such a way that the resulting matrix has the strong COP.
4. If an ordering for the rows of M gives the strong COP, it is called a **COP order** or **COP permutation**.

When the roles of rows and columns are exchanged in Definition 1.3.3, it defines COP on rows. Both are equivalent properties (for the purposes of this thesis) and both conventions are seen the literature. Figure 1.4 gives an example of COP on columns.

M_1: M_1': M_2:

r_1	**1**	0	**1**	r_3	**1**	0	0	d_1	**1**	0	0
r_2	0	**1**	0	r_1	**1**	0	**1**	d_2	**1**	**1**	**1**
r_3	**1**	0	0	r_4	0	**1**	**1**	d_3	0	**1**	0
r_4	0	**1**	**1**	r_2	0	**1**	0	d_4	0	0	**1**

Figure 1.4: Matrices with and without COP. M_1 has COP because by permuting its rows, r_1-r_4, one can obtain M_1' where the **1**s in each column are consecutive. M_2, however, does not have COP since no permutation of its rows, d_1-d_4, will arrange **1**s in each column consecutively [Dom08].

A less restrictive property of binary matrices that is a generalization of the COP is circular-ones property (CROP). The matrix can be visualized to be wrapped around a horizontal cylinder and demands that after some row permutations, if required, in every column the **1**s appear consecutively on the cylinder – note that this implies the 0s also appear consecutively.

Definition 1.3.4 (Circular-ones property (on columns)). Let M be a binary matrix.

1. M has the **strong circular-ones property (strong CROP)** if in every column the **1**s appear consecutively or the 0s appear consecutively or both.
2. M has the **circular-ones property (CROP)** if its rows can be permuted in such a way that the resulting matrix has the strong CROP.
3. If an ordering for the rows of M gives the strong CROP, then it is called the **CROP ordering** or **CROP permutation**.

1.3.2 Sets

Definition 1.3.5 (Set system). Let U be a universe with $|U| = n$.

1. The set $\mathcal{F} \subseteq (2^U \setminus \emptyset)$ is called a **set system** with universe U.
2. Two sets $S, S' \in \mathcal{F}$ are said to **overlap**, denoted by $S \between S'$, if they have a non-empty intersection and neither is contained in the other.

$$S \between S' \text{ if and only if } S \cap S' \neq \emptyset, S \not\subseteq S', S' \not\subseteq S$$

Definition 1.3.6 (Poset). Let $\leq$ be a binary relation on a finite set X. $\leq$ is a **partial order** and $\langle X, \leq \rangle$ is a **poset** if the following hold true for any $a, b, c \in X$.

1. $a \leq a$ (reflexivity)

2. If $a \leq b$ and $b \leq a$ then $a = b$ (antisymmetry)

3. If $a \leq b$ and $b \leq c$ then $a \leq c$ (transitivity)

   An element b **covers** a provided that there exists no third element $x \in X$ such that $a \leq x \leq b$. A directed (or undirected, but with direction implied by vertical positioning of nodes) graph that represents $\leq$ in terms of its covering relation is called the **Hasse diagram** of the poset.

Definition 1.3.7. Let X be a partially ordered set with $\leq$ being the partial order on X.

1. An element $x_m \in X$ is a **maximal upper bound** of X if $\nexists a \in X$ such that $x_m \neq a$ and $x_m \leq a$. A maximal upper bound on X is denoted by $mub(X)$.

2. A set of sets form a **single inclusion chain** when the Hasse diagram of their subset relation forms a single chain.

1.3.3 Graphs

Definition 1.3.8 (Graph, Tree, Path, Graph Isomorphism).

1. An **(undirected) graph** is $G = (V, E)$ is such that V is a finite set and E is a set of unordered pairs, $E \subseteq V \times V$. An element in V is called a **vertex (pl. vertices)** and an element from E is called an **edge**. V and E may be written as $V(G)$ and $E(G)$ respectively. If $u, v \in V$ and $(u, v) \in E$ then v is said to be **adjacent** to u and vice versa. A **subgraph** of G is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$.

2. A **path** is a graph $P = (V, E)$ with vertex set $V = \{v_1, \ldots, v_n\}$ and edge set $E = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-2}, v_{n-1}), (v_{n-1}, v_n)\}$. The vertices v_1, v_n are called the **endpoints** of P. The graph P is called a **cycle** if it has an additional edge (v_n, v_1).

3. A **tree** is a connected graph T that has no subgraph that is a cycle.

4. A **graph isomorphism** between two graphs G_1 and G_2 is a bijection $\phi : V(G_1) \to V(G_2)$ such that $u, v \in V(G_1)$ are adjacent in G_1 if and only if $\phi(u), \phi(v)$ are adjacent in G_2. If such a bijection exists, it is said that G_1 is **isomorphic** to G_2, denoted by $G_1 \cong G_2$.

Definition 1.3.9 (Intersection graph). Let $\mathcal{F}$ be a set system. Then its **intersection graph** $\mathbb{I}(\mathcal{F})$ is a graph such that its vertex set has a bijection to $\mathcal{F}$ and there exists an edge between two vertices if and only if their corresponding sets in $\mathcal{F}$ have a non-empty intersection.

Definition 1.3.10 (Interval graph, Path graph). Let G be a graph.

1. G is an **interval graph** if there exists a set of intervals (of integers) $\mathcal{I}$ such that G is isomorphic to the intersection graph of $\mathcal{I}$.

$$G \cong \mathbb{I}(\mathcal{I})$$

2. G is a **path graph** if there exists a (undirected unrooted) tree T and a set system of paths $\mathcal{P}$ from T such that G is isomorphic to the intersection graph of $\mathcal{P}$.

$$G \cong \mathbb{I}(\mathcal{P})$$

   Path graphs are also known as UV-graphs (Undirected tree where paths intersect in a Vertex) [MW86].

3. G is a **chordal graph** if it has no induced cycle of length greater than 3. Moreover, a chordal graph is characterized as an intersection graph of subtrees of a tree.

The set of interval graphs is a subset of the set of path graphs which is a subset of the set of chordal graphs. In fact, there is another class between chordal graphs and path graphs called RDV-graphs (Rooted Directed tree where paths intersect in a Vertex) [MW86] which is defined exactly as a path graph except that the trees are rooted and directed. There are other variants of path graphs studied in the literature with unrooted directed trees or where intersection of paths are in edges rather than vertices (DV, UE, DE, RDE) [GM03].

1.4 Consecutive-ones Property - a Brief Survey

In this section, a brief survey of the consecutive-ones problem and its optimization problems is presented.

1.4.1 Matrices with COP

As seen earlier, the interval assignment problem (illustrated as the course scheduling problem in Section 1.2), is a special case of the problem we address in this thesis, namely the tree path labeling problem (illustrated as the study group accommodation problem). The interval assignment problem and COP problem are equivalent problems. In this section we will see some of the results that exists in the literature today towards solving the COP problem and optimization problems surrounding it.

Recall that a matrix with COP is one whose rows (columns) can be rearranged so that the **1**s in every column (row) are in consecutive rows (columns). COP in binary

matrices has several practical applications in diverse fields including scheduling [HL06], information retrieval [Kou77] and computational biology [ABH98]. Further, it is a tool in graph theory [Gol04] for interval graph recognition, characterization of Hamiltonian graphs, planarity testing [BL76] and in integer linear programming [HT02, HL06].

The obvious first questions after being introduced to the consecutive ones property of binary matrices are if COP can be detected efficiently in a binary matrix and if so, can the COP permutation of the matrix also be computed efficiently? Recognition of COP in a binary matrix is polynomial time solvable and the first such algorithm was given by [FG65]. A landmark result came a few years later when [Tuc72] discovered the families of forbidden submatrices that prevent a matrix from having COP and most, if not all, results that came later were based on this discovery which connected COP in binary matrices to convex bipartite graphs. In fact, the forbidden submatrices came as a corollary to the discovery that convex bipartite graphs are AT-free on at least one of the vertex partitions in [Tuc72]. The first linear time algorithm for COP testing (COT) was invented by [BL76] using a data structure called PQ-trees. Since then several COT algorithms have been invented – some of which involved variations of PQ-trees [MM96, Hsu01, McC04], some involved set theory and ICPIA [Hsu02, NS09], parallel COT algorithms[AS95, BS03, CY91] and certifying algorithms[McC04].

The construction of PQ-trees in [BL76] draws on the close relationship of matrices with COP to interval graphs. A PQ-tree of a matrix is one that stores all row (column) permutations of the matrix that give the COP orders (there could be multiple orders of rows or columns) of the matrix. This is constructed using an elaborate linear time procedure and is also a test for planarity. PQR-tree is a generalized data structure based on PQ-trees [MM96, MPT98]. [TM05] describes an improved algorithm to build PQR-trees. [Hsu02] describes the simpler algorithm for COT. Hsu also invented PC-trees [Hsu01] which is claimed to be much easier to implement. [NS09] describes a characterization of consecutive-ones property solely based on the cardinality properties of the set representations of the columns (rows); every column (row) is equivalent to a set that has the row (column) indices of the rows (columns) that have one entries in this column (row). This is interesting and relevant, especially to this thesis because it simplifies COT to a great degree.

[McC04] describes a different approach to COT. While all previous COT algorithms gave the COP order if the matrix has the property but exited stating negative if otherwise, this algorithm gives an evidence by way of a certificate even when it has no COP. This enables a user to verify the algorithm's result even when the answer is negative. This is significant from an implementation perspective because automated program verification is hard and manual verification is more viable. Hence having a certificate reinforces an implementation's credibility. Note that when the matrix *has* COP, the COP order is the certificate. The internal machinery of this algorithm is related to the weighted

betweenness problem addressed in [COR98].

1.4.2 Optimization problems in COP

So far we have been concerned about matrices that have the consecutive ones property. However in real life applications, it is rare that data sets represented by binary matrices have COP, primarily due to the noisy nature of data available. At the same time, COP is not arbitrary and is a desirable property in practical data representation [COR98, JKC+04, Kou77]. In this context, there are several interesting problems when a matrix does not have COP but is "close" to having COP or is allowed to be altered to have COP. These are the optimization problems related to a matrix which does not have COP. Some of the significant problems are surveyed in this section.

[Tuc72] showed that a matrix that does not have COP have certain substructures that prevent it from having COP. Tucker classified these forbidden substructures into five classes of submatrices. This result is presented in the context of convex bipartite graphs which [Tuc72] proved to be AT-free in one of the partitions. By definition, convex bipartite graph have half adjacency matrices that have COP on either rows or columns (graph is biconvex if it has COP on both)[Dom08]. A half adjacency matrix is a binary matrix representing a bipartite graph as follows. The set of rows and the set of columns form the two partitions of the graph. Each row node is adjacent to those nodes that represent the columns that have **1**s in the corresponding row. [Tuc72] proves that this bipartite graph has no asteroidal triple in vertex partition corresponding to rows if and only if the matrix has COP on columns and goes on to identify the forbidden substructures for these bipartite graphs. The matrices corresponding to these substructures are the forbidden submatrices.

Once a matrix has been detected to not have COP (using any of the COT algorithms mentioned earlier), it is naturally of interest to find out the smallest forbidden substructure (in terms of number of rows and/or columns and/or number of entries that are **1**s). [Dom08] discusses a couple of algorithms which are efficient if the number of **1**s in a row is small. This is of significance in the case of sparse matrices where this number is much lesser than the number of columns. $(*, \Delta)$-*matrices* are matrices with no restriction on number of **1**s in any column but have at most Δ **1**s in any row. Min COS-R (Min COS-C), Max COS-R (Max COS-C) are similar problems which deals with inducing COP on a matrix. In Min COS-R (Min COS-C) the question is to find the minimum number of rows (columns) that must be deleted to result in a matrix with COP. In the dual problem Max COS-R (Max COS-C) the search is for the maximum number of rows (columns) that induces a submatrix with COP. Given a matrix M with no COP, [Boo75] shows that finding a submatrix M' with all columns but a maximum cardinality subset of rows such that M' has COP is NP complete. [GH02] corrects an error of the

abridged proof of this reduction as given in [GJ79]. [Dom08] discusses all these problems in detail giving an extensive survey of the previously existing results which are almost exhaustively all approximation results and hardness results. Taking this further, [Dom08] presents new results in the area of parameterized algorithms for this problem.

Another problem is to find the minimum number of entries in the matrix that can be toggled to result in a matrix with COP. [Vel85] discusses approximation of COP AUGMENTATION which is the problem of changing of the minimum number of zero entries to **1**s so that the resulting matrix has COP. This problem is known to be NP complete due to [Boo75]. [Vel82] proves, using a reduction to the longest path problem, that finding a Tucker's forbidden submatrix of at least k rows is NP complete.

[JKC+04] discusses the use of matrices with almost-COP (instead of one block of consecutive **1**s, they have x blocks, or *runs*, of consecutive **1**s and x is not too large) in the storage of very large databases. The problem is that of reordering of a binary matrix such that the resulting matrix has at most k runs of **1**s. This is proved to be NP hard using a reduction from the Hamiltonian path problem.

1.5 Generalization of COP - the Motivation

Section 1.4.1 introduced a succinct characterization for consecutive-ones property which is solely based on the cardinality properties of the set representations of the matrix's columns [NS09]. This result is very relevant to this thesis because aside from it simplifying COT to a great degree, our generalization problem is motivated by their results.

[NS09] characterizes interval assignments to the sets which can be obtained from a single permutation of the rows. For an assignment to be feasible, the cardinality of the interval assigned to each set in the system must be same as the cardinality of the set, and the intersection cardinality of any two intervals must be same as the intersection cardinality of their corresponding sets. While this is obviously a necessary condition, this result shows this is also sufficient. [NS09] calls this an Intersection Cardinality Preserving Interval Assignment (ICPIA). This paper generalizes the idea from [Hsu02] of decomposing a given binary matrix into prime matrices for COT and describes an algorithm to test if an ICPIA exists for a given set system.

The equivalence of the problem of testing for the consecutive-ones property to the constraint satisfaction problem of interval assignment [NS09] or interval labeling [KKLV10] is as follows. Every column (row) of the binary matrix can be converted into a set of non-negative integers which are the indices of rows (columns) with **1**s in that column (row). It is apparent that if the matrix has COP in columns (rows), then constructing such sets after applying the COP permutation to the rows (columns) of the matrix will

result in sets with consecutive integers. In other words, after application of COP reordering, the sets are intervals. Indeed the problem now becomes finding interval assignments to a given set system such that there exists a permutation of the universe of set of row indices (column indices) which converts each set to its assigned interval.

The problem of interest in this thesis, namely, tree path labeling problem, is a natural generalization of the interval assignment problem or the COT problem. The problem is defined as follows – given a set system $\mathcal{F}$ from a universe U and a target tree T, does there exist a bijection from U to the vertices of T such that each set in the system maps to a path in T. We refer to this as the Compute Feasible Tree Path Labeling problem [1] or simply *tree path labeling* problem for an input set system and target tree pair – $(\mathcal{F}, T)$. The special case of the target tree being a path, is the interval assignment problem. We focus on generalizing the notion of an ICPIA [NS09] to characterize feasible path assignments. We show that for a given set system $\mathcal{F}$, a tree T, and an assignment of paths from T to the sets, there is a feasible [2] bijection between U and $V(T)$ if and only if the intersection cardinalities among any three sets (not necessarily distinct) is equal to that of the corresponding paths assigned to them and the input passes a filtering algorithm (described in this thesis) successfully [3]. This algorithmic characterization gives a natural data structure that stores all the feasible bijections between U and $V(T)$. This reduces the search space for the solution considerably from the universe of all possible bijections between U and $V(T)$ to only those bijections that maintain the characterization. Further, the filtering algorithm is also an efficient algorithm to test if a tree path labeling [4] is feasible.

1.6 Summary of New Results in this Thesis

We see in Section 1.5 that pairwise intersection cardinality preservation is necessary and sufficient for an interval assignment to be feasible for a given hypergraph [5] and thus is a characterization for COP [NS09]. In our work we extend this characterization and find that trio-wise intersection cardinality preservation makes a tree path labeling [6](TPL) feasible, which is a generalization of the COP problem. This problem is defined by Feasible Tree Path Labeling (see Section 3.1 for problem definition).

1. Problem is formally defined in Section 3.1.
2. The notion of *feasibility* is formally defined in Section 3.2.
3. Formally defined by Theorem 3.3.13
4. The terms *tree path labeling* and *tree path assignment* are, in informal language, synonyms. Formally, the former refers to the bijection $\ell : \mathcal{F} \to \mathcal{P}$. The latter refers to the set of ordered pairs $\{(S, P) \mid S \in \mathcal{F}, P \in \mathcal{P}\}$. $\mathcal{P}$ is a set of paths on T.
5. A *hypergraph* is an alternate representation of a set system and will be used in this thesis, in line with [KKLV10] (see Section 3.2 for the formal definition).
6. A *tree path labeling* ℓ is a bijection of paths from the target tree T to the hyperedges in given hypergraph $\mathcal{F}$ (see Section 3.2 for the formal definition).

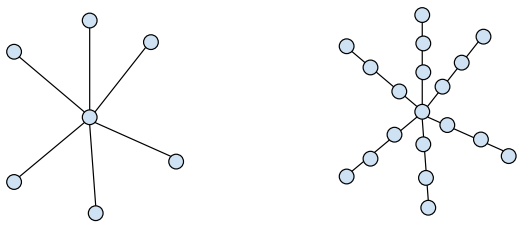


Figure 1.5: Examples of k-subdivided stars. (a) $k = 0$ (b) $k = 2$

We give a necessary and sufficient condition by way of *Intersection Cardinality Preservation Path Labeling* (ICPPL) and a filtering algorithm for Feasible Tree Path Labeling to output in affirmative. ICPPL captures the trio-wise cardinality property described earlier (see Section 3.3 for the definition of ICPPL.). This characterization can be checked in polynomial time. A relevant consequence of this constructive procedure is that it is sufficient to iteratively check if three-way intersection cardinalities are preserved. In other words, in each iteration, it is sufficient to check if the intersection of any three hyperedges is of the same cardinality as the intersection of the corresponding paths. Thus this generalizes the well studied question of the feasible interval assignment problem which is the special case when the target tree T is simply a path [Hsu02, NS09].

Aside from checking if a given TPL is feasible, we also solve the problem of computing a feasible TPL for a given hypergraph and target tree, if one exists. This problem is the Compute Feasible Tree Path Labeling problem (see Section 3.1 for problem definition).

We present a polynomial time algorithm for Compute Feasible Tree Path Labeling when the target tree T belongs to a special class of trees called *k-subdivided stars* (see Definition 3.2.6 for the formal definition) and when the hyperedges in the hypergraph $\mathcal{F}$ have at most $k + 2$ vertices (Compute k-subdivided Star Path Labeling– see Section 3.1 for problem definition). A couple of examples of k-subdivided stars are given in Figure 1.5.

In spite of this being a restricted case, we believe that our results are of significant interest in understanding the nature of Graph Isomorphism which is polynomial time solvable in interval graphs while being hard on path graphs[KKLV10]. k-subdivided stars are a class of trees which are in many ways very close to intervals or paths. Each ray [7] are independent except for the root [8] and hence can be considered as an independent interval till the root. Our algorithm builds on this fact and uses the interval assignment algorithm[NS09] up until "reaching" the root and then uses the trio-wise intersection cardinality (the extra condition in ICPPL that generalizes ICPIA) check to resolve the

7. The path from a leaf to the root, the vertex with highest degree, is called a *ray* of the k-subdivided star (see Section 3.2 for the formal definition).

8. The vertex with maximum degree in a k-subdivided star is called *root* (see Section 3.2 for the formal definition).

ambiguity about which ray the algorithm should "grow" the solution into in the next iteration.

We also have an algorithm for solving Compute Feasible Tree Path Labeling with no restrictions on the target tree or set cardinality which runs in exponential time. This algorithm finds a path labeling from T by decomposing the problem into subproblems of finding path labeling of subsets of $\mathcal{F}$ from subtrees of T. Given the fact that binary matrices naturally represent a set system (see Section 1.5) and that the *overlap* relation (see Section 3.2 for the formal definition) between the sets involved is an obvious equivalence relation, $\mathcal{F}$ quite naturally partitions into equivalence classes known as *overlap components* (see Section 3.2 for the formal definition). In the context of COP, overlap components were used in [Hsu02] and [KKLV10]. Moreover, [NS09] discovered that these equivalence classes have a certain total order in the case of COP (see Section 3.6 for the formal definition). We extend this to TPL and find that when $\mathcal{F}$ is a path hypergraph [9], the classes can be partially ordered as an in-tree in polynomial time. Once $\mathcal{F}$ is "broken" into overlap components, one must identify the subtree of T that it needs to map to and this is the hard part which is currently open to be solved in polynomial time.

9. If there exists an FTPL for a hypergraph $\mathcal{F}$, it is called a *path hypergraph*.

CHAPTER 2

CONSECUTIVE-ONES PROPERTY – A SURVEY OF IMPORTANT RESULTS

This chapter surveys several results that are significant to this thesis or to COP in general. These predominantly pertain to characterizations of COP, algorithmic tests to check for COP and certificates for binary matrices that do not have COP.

The chapter is organized as follows. Section 2.1 discusses the close connections of matrices with COP in graph theory. Section 2.2 discusses the important results in the area of consecutive-ones property testing (COT) – we describe the history of this problem's several solutions including Tucker's forbidden submatrices, PQ-tree, PQR-tree and ICPIA.

It is worth noting that we make a very interesting observation about PQR-tree, generalised PQ-tree and gPQ-tree – that the theory that all these independent data structures are built upon are very similar and equivalent. This is exhaustively described in Section 2.2.3 and curiously, this comparison has not been encountered by us in the literature.

Section 2.3 briefly touches upon the scenario of algorithmically verifying that a matrix indeed does not have COP. Its significance lies in the fact that almost all COT algorithms, upon detection of no COP, simply reports the same without any certificate for verification during algorithm implementation. The two approaches presented in this section gives a certificate.

2.1 COP in Graph Theory

COP is closely connected to several types of graphs by way of describing certain combinatorial graph properties. There are also certain graphs, like convex bipartite graphs, that are defined solely by one of its associated matrices having COP. In this section we will see the relevance of consecutive-ones property to graphs. To see this we introduce certain binary matrices that are used to define graphs in different ways. While adjacency matrix is perhaps the most commonly used such matrix, Definition 2.1.1 defines this and a few more.

Definition 2.1.1 (Matrices that define graphs [Dom08, Def. 2.4]). Let G and H be defined as follows. $G = (V, E_G)$ is a graph with vertex set $V = \{v_i \mid i \in [n]\}$ and edge set $E_G \subseteq \{(v_i, v_j) \mid i, j \in [n]\}$ such that $|E_G| = m$. $H = (A, B, E_H)$ is a bipartite graph with partitions $A = \{a_i \mid i \in [n_a]\}$ and $B = \{b_i \mid i \in [n_b]\}$.

2.1.1–i. **Adjacency matrix** of G is the symmetric $n \times n$ binary matrix M with $m_{i,j} = \mathbf{1}$ if and only if $(v_i, v_j) \in E_G$ for all $i, j \in [n]$.

2.1.1–ii. **Augmented adjacency matrix** of G is obtained from its adjacency matrix by setting all main diagonal elements to $\mathbf{1}$, i. e. $m_{i,i} = \mathbf{1}$ for all $i \in [n]$.

2.1.1–iii. **Maximal clique matrix** or **vertex-clique incidence matrix** of G is the $n \times k$ binary matrix M with $m_{i,j} = \mathbf{1}$ if and only if $v_i \in C_j$ for all $i \in [n], j \in [k]$ where $\{C_j \mid j \in [k]\}$ is the set of maximal cliques of G.

2.1.1–iv. **Half adjacency matrix** of H is the $n_a \times n_b$ binary matrix M with $m_{i,j} = \mathbf{1}$ if and only if $(a_i, b_j) \in E_H$.

Now we will see in Definition 2.1.2 certain graph classes that is related to COP or CROP.

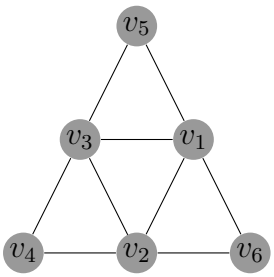
Definition 2.1.2 (Graphs that relate to COP [Dom08, Def. 2.5]). Let G be a graph and H be a bipartite graph.

2.1.2–i. G is **convex-round** if its adjacency matrix has the CROP.

2.1.2–ii. G is **concave-round** if its augmented adjacency matrix has CROP. [BJHY00]

2.1.2–iii. G is an **interval graph** if its vertices can be mapped to intervals on the real line such that two vertices are adjacent if and only if their corresponding intervals overlap [Ben59]. G is an interval graph if and only if its maximal clique matrix has COP [FG65] (This follows [GH64] which states that the maximal cliques of interval

G_1:

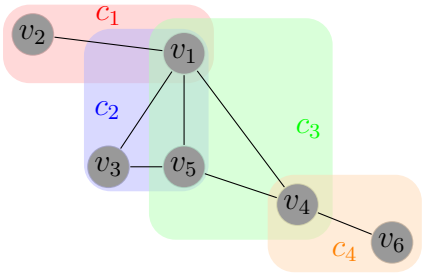


$\mathbf{A_2}$	v_1	v_2	v_3	v_4	v_5	v_6
v_1	**1**	**1**	**1**	0	**1**	**1**
v_2	**1**	**1**	**1**	**1**	0	**1**
v_3	**1**	**1**	**1**	**1**	**1**	0
v_4	0	**1**	**1**	**1**	0	0
v_5	**1**	0	**1**	0	**1**	0
v_6	**1**	**1**	0	0	0	**1**

$\mathbf{A_1}$	v_1	v_2	v_3	v_4	v_5	v_6
v_1	0	**1**	**1**	0	**1**	**1**
v_2	**1**	0	**1**	**1**	0	**1**
v_3	**1**	**1**	0	**1**	**1**	0
v_4	0	**1**	**1**	0	0	0
v_5	**1**	0	**1**	0	0	0
v_6	**1**	**1**	0	0	0	0

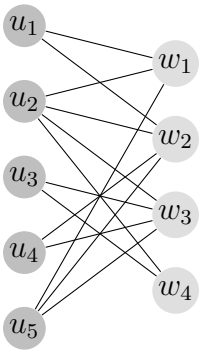
$\mathbf{A'_2}$	v_1	v_6	v_2	v_4	v_3	v_5
v_1	**1**	**1**	**1**	0	**1**	**1**
v_6	**1**	**1**	**1**	0	0	0
v_2	**1**	**1**	**1**	**1**	**1**	0
v_4	0	0	**1**	**1**	**1**	0
v_3	**1**	0	**1**	**1**	**1**	**1**
v_5	**1**	0	0	0	**1**	**1**

G_2:



$\mathbf{B}$	c_1	c_2	c_3	c_4
v_1	**1**	**1**	**1**	0
v_2	**1**	0	0	0
v_3	0	**1**	0	0
v_4	0	0	**1**	**1**
v_5	0	**1**	**1**	0
v_6	0	0	0	**1**

H:



$\mathbf{C}$	w_1	w_2	w_3	w_4
u_1	**1**	**1**	0	0
u_2	**1**	**1**	**1**	**1**
u_3	0	0	**1**	**1**
u_4	0	**1**	**1**	0
u_5	**1**	**1**	**1**	0

Figure 2.1: A_1 is the *adjacency matrix* and A_2 is the *augmented adjacency matrix* of G_1. A'_2 is obtained from A_2 by permuting its rows and columns to achieve *CROP order*, i. e. A_2 has CROP – thus G_1 is *a concave-round graph* (Def. 2.1.2 ii) and *a circular-arc graph* (Tab. 2.1) B is the maximal clique matrix of G_2 and has COP – thus G_2 is an *interval graph* (Def. 2.1.2 ii). C is the half adjacency matrix of bipartite graph H and has COP on rows – thus H is a convex bipartite graph.

graph G can be linearly ordered such that for all $v \in V(G)$, cliques containing v are consecutive in the ordering [Gol04, Th. 8.1].)

a. G is a **unit interval graph** if it is an interval graph such that all intervals have the same length.

b. G is a **proper interval graph** if it is an interval graph such that no interval properly contains another.

The set of unit interval graphs and the set of proper interval graphs coincide [Rob69, Gar07].

2.1.2–iv. G is a **circular-arc graph** if its vertices can be mapped to a set of arcs on a circle such that two vertices are adjacent if and only if their corresponding arcs overlap.

2.1.2–v. H is **convex bipartite on columns (rows)** if its half adjacency matrix has COP on rows (columns).

2.1.2–vi. H is **biconvex bipartite** or **doubly convex**[CY95] if its half adjacency matrix has COP on both rows and columns.

2.1.2–vii. H is **circular convex** if its half adjacency matrix has CROP.

Interval graphs[1] and circular-arc graphs have a long history in research. The interest around them is due to their very desirable property that several problems that are NP-complete on general graphs, like finding a maximum clique or minimum coloring or independent set, are polynomial time solvable in these graph classes [CLRS01]. In a similar fashion, a lot of problems that are hard on general matrices have efficient solutions on matrices with COP or CROP [Dom08, more citations pg. 33].

Table 2.1 summarises the way these graphs are characterized by their matrices having COP or CROP. Our focus in this chapter (and thesis) is mainly COP and having seen how useful COP is in identifying or characterizing many types of graphs, we will now see results that study recognition of COP in matrices in the following section.

2.2 Matrices with COP

The most important questions with respect to a particular property desired in a structure/object are perhaps the following.

1. [McC04] cites that the problem of recognizing interval graphs has significance in molecular biology. Interestingly, in the late 1950s, before the structure of DNA was well-understood, Seymour Benzer was able to show that the intersection graph of a large number of fragments of genetic material was an interval graph [Ben59]. This was regarded as compelling evidence that genetic information was somehow arranged inside a structure that had a linear topology which we now know to be true from the discovery of linear structure of DNA.

Graph class	Adjacency matrix	Augmented adjacency matrix	Half adjacency matrix	Maximal clique matrix
Convex-round	⇔ CROP (by defn.)			
Concave-round ⋂		⇔ CROP (by defn.)		
Circular-arc ⋃		⇐ CROP [Tuc72]		⇐ CROP
Helly circular-arc ⋃		⇐ COP		⇔ CROP [Gav74a]
Interval ⋃		⇐ COP		⇔ COP [FG65]
Proper/unit interval		⇔ COP [Rob69]		⇔ COP $r+c$ [Fis85]
		Bipartite graphs		
Circular convex ⋃			⇔ CROP (by defn.)	
Convex ⋃			⇔ COP (by defn.)	
Biconvex			⇔ COP $r+c$ (by defn.)	

Table 2.1: Relationship between graph classes and graph matrices in terms of their COP or CROP. The arrows indicate the implication of the statement. ⇔ indicates that the membership in the graph class and the matrix property are equivalent. ⇐ indicates that the matrix property implies the membership in the graph class. $r+c$ indicates that the property holds on rows and columns of the matrix. ⋃ and ⋂ indicate that the graph class above is a superset or subset, respectively, of the graph class below. [Dom08, Tab. 2.1]

- Does the desired property exist in the given input?
- If the test is affirmative, what is a certificate of the affirmative?
- If the test is negative, what are the optimization possibilities for the property in the input? In other words, how close to having the property can the input be?
- If the test is negative, what is a certificate of the negative?

In this section and the rest of the chapter we see results that shaped the corresponding areas respectively for consecutive-ones property in binary matrices.

a. Does a given binary matrix have COP?

b. What is the COP permutation for the given matrix with COP?

c. What are the optimizations possible and practically useful on the given matrix without COP?

d. If algorithm for (a) returns **false**, can a certificate for this be computed?

Without doubt, besides computing answers to these questions, we are interested in the efficiency of these computations in terms of computational complexity theory. Results towards questions (a) and (b) are surveyed in this section. Those for question (d) is discussed in Section 2.3. Question (c) is discussed in brief in Section 1.4.2 – it is not elaborated in this chapter since the focus of this thesis is characterization and recognition rather than optimization.

One way to design an algorithm to test for COP is by deriving one from any interval graph recognition algorithm using the result [HMPV00] [Dom08, Th 2.7], restated below in Theorem 2.2.1, which demonstrates how such a derivation can be done. They use something called *Lex-BFS ordering* of the vertices of a graph to decide in linear time whether the graph is an interval graph. They also show how any interval graph recognition algorithm can be used to recognize matrices with COP.

Theorem 2.2.1 ([HMPV00, Th. 2]). *For a binary matrix M the following statements are equivalent.*

1. *The row adjacency graph $G_r(M)$ is an interval graph and M is its maximal clique matrix.*
2. *The columns of M are maximal and M has the COP for rows.*

However, this does not necessarily yield an efficient algorithm. We will see results that directly solve the problem of COT on matrices since it is known that questions (a) and (b) stated above for COP are efficiently solvable. Table 2.2 gives a snapshot of these results.

1899	First mention of COP (archaeology)	[Ken69]
1951	Heuristics for COT	[Rob51]
1965	First polynomial time algorithm for COP testing	[FG65]
1972	Characterization for COP– forbidden submatrices	[Tuc72]
1976	First linear time algorithm for COT – PQ-tree	[BL76]
1992	Linear time algorithm COT without PQ-tree	[Hsu02]
2001	PC-tree – a simplification of PQ-tree	[Hsu01, HM03]
1996	PQR-tree – generalization of PQ-tree for any binary matrix regardless of its COP status	[MM96]
1998	Almost linear time to construct PQR-tree	[MPT98]
2004	A certifying algorithm for no COP. Generalized PQ-tree.	[McC04]
2009	Set theoretic, cardinality based characterization of COP – ICPIA	[NS09]
2010	Logspace COP testing	[KKLV10]

Table 2.2: A brief history of COP research

The first polynomial time algorithm for COP testing was by [FG65] which uses overlapping properties of columns with **1**s. Their result has close relations to the characterization of interval graphs by [GH64]. A graph G is an interval graph if and only if all its maximal cliques can be linearly ordered such that for any vertex v in G, all the cliques that v is incident on are consecutive in this order. Clearly, this means that the maximal clique incidence matrix (see Definition 2.1.1 iii) must have COP on rows.

A few years later, a deeply significant result based on very different ideas in understanding COP came from Tucker which gave a combinatorial (negative) characterization of matrices with COP [Tuc72]. This result influenced most of the COP results that followed in the literature including linear time algorithms for COP recognition.

2.2.1 Tucker's forbidden submatrices for COP

[Tuc72] discovered certain forbidden structures for convex bipartite graphs [2] and by definition of this graph class, this translates to a set of forbidden submatrices for matrices with consecutive-ones property. The following are the theorems from [Tuc72] that achieved this characterization.

Theorem 2.2.2 states that convex bipartite graphs cannot have *asteroidal triples* (see Definition 2.2.1) contained in the corresponding vertex partition. This partition of the bipartite graph corresponds to columns (rows) if its half adjacency matrix has COP columns (rows). Theorem 2.2.3 lists the structures in a bipartite graph that force one of its vertex partitions to have asteroidal triples – in other words, it identifies the subgraphs that prevent the graph from being convex bipartite.

Definition 2.2.1 (Asteroidal triple). An **asteroidal triple** in a graph G is a tuple of vertices $\{v_1, v_2, v_3\} \subset V$ such that between every pair of vertices from this tuple there exists

2. The terminology in [Tuc72] differs. It uses the term *graphs with V_1-consecutive arrangement* instead of *convex bipartite graphs*.

a path that avoids the closed neighbourhood of the third vertex.

Theorem 2.2.2 ([Tuc72, Th. 6], [Dom08, Th. 2.3]). *A bipartite graph $G = (V_1, V_2, E)$ is convex bipartite on columns*[3] *if and only if V_1 contains no asteroidal triple of G.*

Theorem 2.2.3 ([Tuc72, Th. 7], [Dom08, Th. 2.4]). *In a bipartite graph $G = (V_1, V_2, E)$ the vertex set V_1 contains no asteroidal triple if and only if G contains none of the graphs G_{I_k}, G_{II_k}, G_{III_k} (with $k \geq 1$), G_{IV}, G_V as shown in Figure 2.2 as subgraphs.*

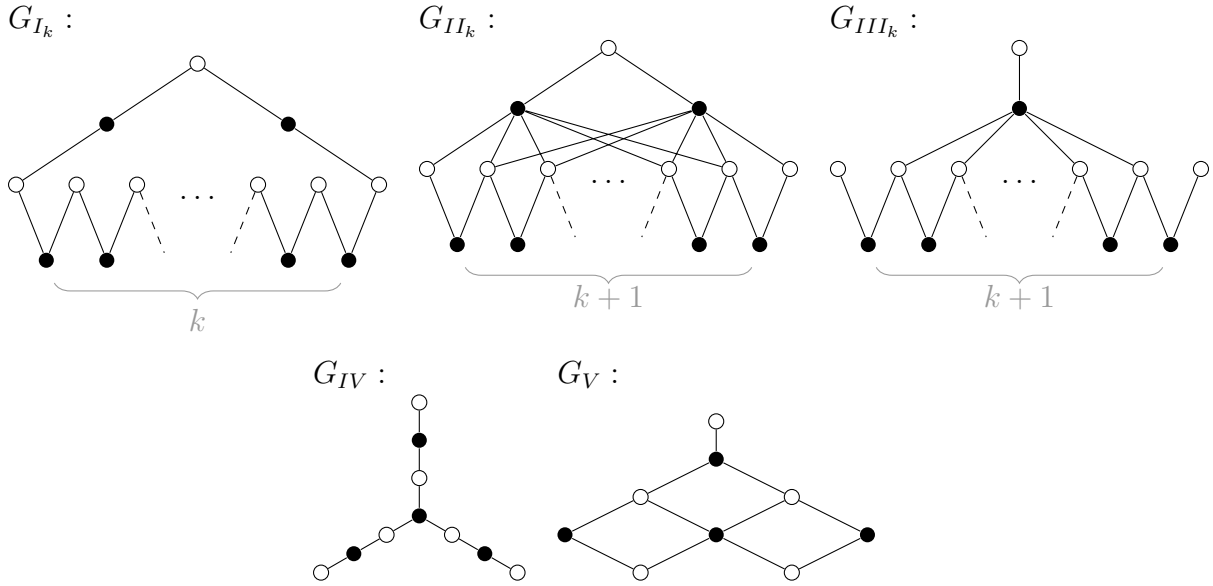


Figure 2.2: Tucker's forbidden subgraphs for convex bipartite graphs.

Theorem 2.2.2 and Theorem 2.2.3 result in the following Theorem 2.2.4 which characterizes matrices with COP.

Theorem 2.2.4 ([Tuc72, Th. 9], [Dom08, Th. 2.5]). *A matrix M has COP if and only if it contains none of the matrices M_{I_k}, M_{II_k}, M_{III_k} (with $k \geq 1$), M_{IV}, M_V as shown in Figure 2.3 as submatrices.*

It can be verified that the matrices in Figure 2.3 are the half adjacency matrices of the graphs in Figure 2.2 respectively which is not surprising due to Definition 2.1.2 v.

2.2.2 Booth and Lueker's PQ-tree – a linear COT algorithm

Booth and Lueker in their paper [BL76] gave the first linear algorithm for consecutive-ones property testing while given a linear time interval graph recognition algorithm by a simplification of [ALC67]'s planarity test algorithm. This COT algorithm has time complexity $O(m + n + f)$ where $m \times n$ is the order of the input matrix and f is the number of **1**s in it. [BL76] introduces a data structure called PQ-tree and their COP testing algorithm is a constructive one that outputs a PQ-tree if the input has COP. A

3. Abridged to match terminology adopted in this document. See previous note.

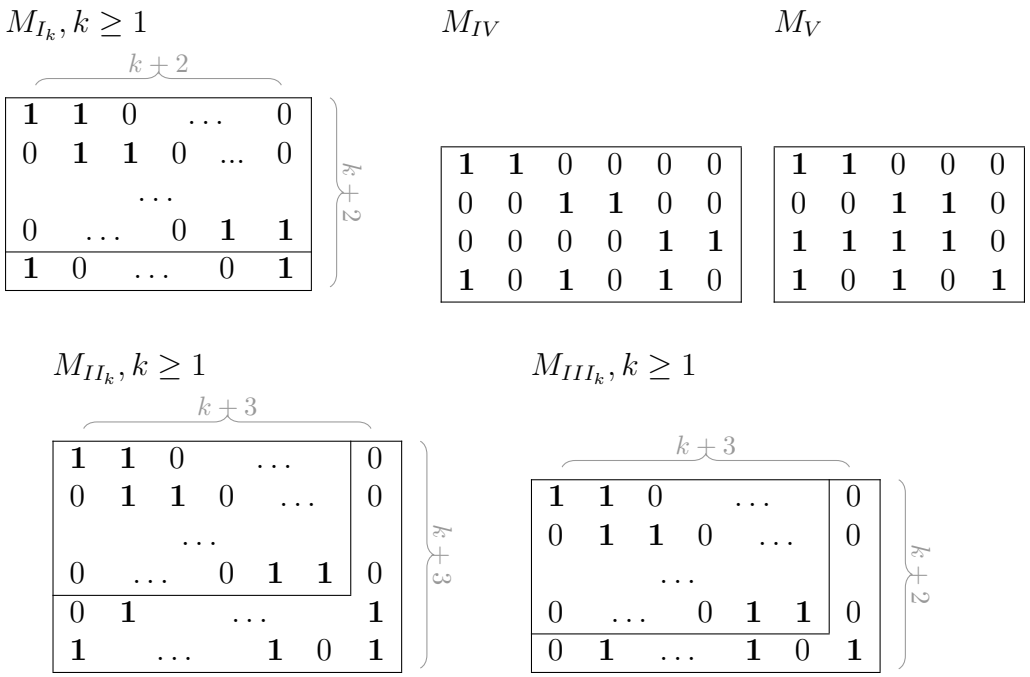


Figure 2.3: Tucker's forbidden submatrices for matrices with COP. [Tuc72]

PQ-tree represents all the COP orderings of the matrix it is associated with. [BL76]'s algorithm uses the fact that if a matrix has COP, a PQ-tree for it can be constructed. It is interesting to note that aside from interval graph recognition and COP testing, PQ-tree is also useful in other applications like finding planar embeddings of planar graphs [ALC67, McC04] and recognizing CROP in a matrix.

Definition 2.2.2 (PQ-tree [BL76, McC04]). A PQ-tree of matrix M with COP on columns (rows), is a tree with the following properties.

i. Each leaf uniquely represents a row (column) of M. The leaf order of the tree gives a COP order for column (row) (COP order for column requires permutation of rows and vice versa.) for M.

ii. Every non-leaf node in the tree is labeled P or Q.

iii. The children of P nodes are unordered. They can be permuted in any fashion to obtain a new COP order for M.

iv. The children of Q nodes are linearly ordered. Their order can be reversed to obtain a new COP order for M.

See Figure 2.4 for an example of PQ-tree. It may be noted that there is no way an empty set of COP orderings can be represented in this data structure. For this reason, PQ-tree is undefined for matrices that do not have COP. Thus effectively, there exists a bijection between set of matrices with COP and the set of PQ-trees (accurately speaking, each matrix with COP bijectively maps to an equivalence class of PQ-trees resulting from properties (iii) and (iv)).

r_1	**1**	0	**1**	0	0	0	0
r_2	**1**	0	**1**	0	0	0	0
r_3	**1**	0	0	**1**	0	0	0
r_4	**1**	0	0	**1**	0	0	0
r_5	**1**	0	0	0	0	0	0
r_6	**1**	**1**	0	0	0	0	0
r_7	0	**1**	0	0	0	**1**	0
r_8	0	**1**	0	0	**1**	**1**	0
r_9	0	**1**	0	0	**1**	**1**	0
r_{10}	0	**1**	0	0	**1**	**1**	**1**
r_{11}	0	**1**	0	0	**1**	0	**1**

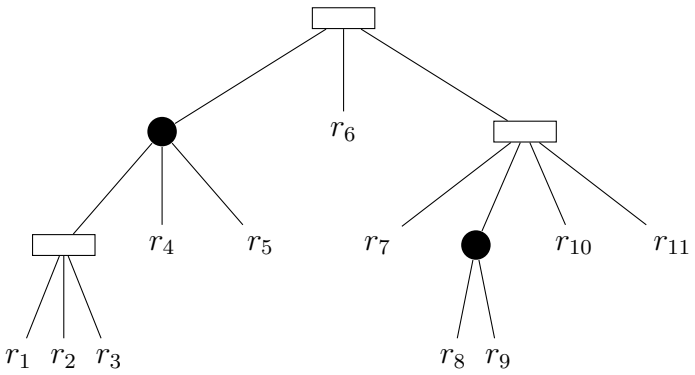


Figure 2.4: An example for PQ-tree with the associated matrix with COP. The dark circular nodes are P-nodes and rectangular nodes are Q-nodes. The root is the topmost node which in this case is a Q-node. Permuting the order of the left child of the root, which is a P-node, we see that $(r_4, r_1, r_2, r_3, r_5, r_6, r_7, r_8, r_9, r_{10}, r_{11})$ is a COP order. Reversing the order of the right child of the root, which is a Q-node, we see that $(r_1, r_2, r_3, r_4, r_5, r_6, r_{11}, r_{10}, r_8, r_9, r_7)$ is yet another COP order. [McC04, Fig. 1]

The [BL76] algorithm with input $n \times m$ matrix M starts with a PQ-tree for a vacuous $n \times 0$ matrix M' (submatrix induced by 0 columns). This is known as a *universal* PQ-tree which is one with its root as a P node and only leaves as its children – each leaf representative of a row of input (by definition of COP for columns). This induced submatrix M' vacuously has COP. Each column is then added iteratively to M' to check if the new M' has COP. By a complicated, but linear, procedure the algorithm does one of the following actions in each iteration: (a) declare that M has no COP, or (b) modify the current PQ-tree to represent the new M' (which clearly, must have COP, since if not, option (a) would have been executed).

Judging from notes in literature, this algorithm is apparently notoriously difficult to program. In the procedure to modify the PQ-tree at each iteration, nodes are considered from leaves to the root. At each node considered, it uses one of nine templates to determine how the tree must be altered in the vicinity of this node. Recognition of this template poses a difficult challenge in terms of implementing it. Each template is actually a representative of a larger class of similar templates, which must be dealt with explicitly by a program[McC04].

After the invention of PQ-trees, presumably due to the implementation challenge it posed, there has been several variants of the same in the literature, like PC-tree [SH99, Hsu01, HM03], generalized PQ-tree [KM89, McC04], PQR-tree [MM96, MPT98] etc. Most of these are generalizations of PQ-tree – for instance, PC-tree is generalized to matrices with CROP, PQR-tree and generalized PQ-tree are generalized to matrices and set systems with or without COP. [KM89] invented a modified form of PQ-tree, called MPQ-trees, a simpler incremental update of the tree only for recognizing interval graphs. [KR88] constructed efficient parallel algorithms for manipulating PQ-trees.

In the next few sections we will see some of these variations.

2.2.3 PQR-tree – COT for set systems

Section 1.5 mentions how a binary matrix naturally maps to a system of sets. A set can be constructed for each column of matrix with its elements being those row indices at which the column has **1**s. Thus the collection of sets corresponding each column of the matrix forms a set system with universe as the set of all row indices of the matrix. This simple construction is formally described in Definition 2.2.3 along with the idea of consecutive-ones property for set systems (As seen in Section 1.2.1).

Definition 2.2.3 (Consecutive-ones property for set systems). Let M be a binary matrix of order $n \times m$ and $\{c_i \mid i \in [m]\}$ be the columns in M. A set system $\mathcal{F}_M = \{S_i \mid S_i \subseteq [n], i \in [m]\}$ is defined such that for every column c_i of M, set $S_i = \{j \mid m_{ji} = \mathbf{1}\}$. The collection $\mathcal{F}_M$ is the **set system of binary matrix** M. The **binary matrix for set system** $\mathcal{F}$ is conversely constructed and denoted by $M^{\mathcal{F}}$. Thus, $M^{\mathcal{F}_M} = M$.

A set system $\mathcal{F}$ from universe U, $|U| = n$ has the **consecutive-ones property** if there exists a linear order or permutation $\sigma = w_1 w_2 \dots w_n$ that can be applied to U such that each set $S \in \mathcal{F}$ becomes a consecutive subsequence (also termed an **interval**) $w_i w_{i+1} \dots w_{i+k-1}$ on σ for some positive integer $i \leq n + 1 - k$ where $k = |S|$.

The set $\texttt{valid}\,(U, \mathcal{F})$ represents all COP orders of $\mathcal{F}$ in U.

It is easy to see the equivalence of this definition to COP for matrices in Definition 1.3.3.

Before proceeding to describe PQR-tree per se, we will see a few more terminologies that will make the subsequent discussion in this section simpler.

Definition 2.2.4 (Orthogonal sets [Nov89, MM96, McC04] [4] etc.). Let $\mathcal{F}$ be a set system with universe U and sets $A, B \in \mathcal{F}$.

1. A and B are said to have a **trivial intersection** if $A \cap B$ is $\emptyset$ or A or B. In other words, A and B are either disjoint or one is the subset of the other.
2. A and B are called **mutually orthogonal** or A is **orthogonal to** B and vice versa, if they have a trivial intersection.
3. **Trivial subsets** of a universe U, denoted by $\mathcal{T}(U)$, are sets that have trivial intersections with any set in 2^U. These sets are U, singleton sets in 2^U and $\emptyset$ [Nov89, MM96]. Thus, $\mathcal{T}(U) = \{U\} \bigcup \{\{v\} \mid v \in U\} \bigcup \{\emptyset\}$.
4. *Orthogonal sets of a set system* [5] $\mathcal{F}$ with universe U are subsets of U that are orthogonal to all sets in $\mathcal{F}$. The set of all orthogonal sets to $\mathcal{F}$ is denoted by $\mathcal{F}^{\perp}$.

4. [McC04] does not use the term "mutually orthogonal" but refers to the same idea as "sets that do not overlap". This terminology is also used in other literature like [NS09, Hsu02].

5. [McC04, Def. 3.1] uses the term **non-overlapping family** of $\mathcal{F}$ and denotes it by $\mathcal{N}(\mathcal{F})$.

5. $\mathcal{F}$ is called **complete**[6] if the following hold true for every pair of non-orthogonal (overlapping) sets A, B in $\mathcal{F}$.

   (i) $\mathcal{T}(U) \subset \mathcal{F}$ (ii) $A \cup B \in \mathcal{F}$ (iii) $A \cap B \in \mathcal{F}$ (iv) $A \setminus B \in \mathcal{F}$ (v) $B \setminus A \in \mathcal{F}$

   In other words, $\mathcal{F}$ contains all the trivial subsets of U, $A \cup B$ and the partitions of $A \cup B$ defined by intersection and set difference.

6. $\overline{\mathcal{F}}$ represents the smallest super set system of $\mathcal{F}$ that is complete[7]

7. The binary operator / is defined as follows. Let A, B be two sets from set system $\mathcal{F}$.

$$A/B = A \setminus B \cup \{x_B\}$$
$$\mathcal{F}/B = \{S/B \mid S \in \mathcal{F}, S \nsubseteq B\}$$

   $x_B \notin U$ is a new element introduced as shown above as a representative for B after removing elements of B.

An important observation made by [MM96] is presented now along with a few theorems that help in decomposing the COP problem on $\mathcal{F}$ into subproblems.

Observation 2.2.1 ***([MM96, Sec. 3])***. If $\mathcal{F}$ is a set system with COP then, after applying the COP order, not only must every set in $\mathcal{F}$ be consecutive but the following sets must also be consecutive for any $A, B \in \mathcal{F}$.

1. The intersection $A \cap B$
2. The union $A \cup B$ if $A \cap B \neq \emptyset$
3. The relative complements $A \setminus B$ and $B \setminus A$ if $B \nsubseteq A$ and $A \nsubseteq B$ respectively.
4. Also note that trivially, sets in $\mathcal{T}(U)$ are consecutive in any permutation of U ($\emptyset$ is considered consecutive by convention.).

The following theorem gives a very interesting property of orthogonal sets and $\overline{\mathcal{F}}$.

Theorem 2.2.5 ([MM96, Th. 3,6]). *For any set system $\mathcal{F}$ we have the following.*

$$\texttt{valid}(\mathcal{F}) = \texttt{valid}\left(\overline{\mathcal{F}}\right)$$
$$\mathcal{F}^{\perp} = \overline{\mathcal{F}^{\perp}} = (\overline{\mathcal{F}})^{\perp}$$

Proposition 2.2.6. Any set that is consecutive on all the COP permutations of $\mathcal{F}$ is present in $\overline{\mathcal{F}}$.

6. [McC04] calls this a **weakly partitive family**.
7. [McC04, Def. 3.2] calls this the **weak closure** of $\mathcal{F}$ denoted by $\mathcal{W}(\mathcal{F})$.

Proposition 2.2.6 is owing to the following theorem by which [MM96] describes a way to decompose the problem of finding all COP orders of $\mathcal{F}$ into two subproblems using sets in $\overline{\mathcal{F}} \cap \mathcal{F}^{\perp}$. The operator / used is defined in Definition 2.2.4 (7).

Theorem 2.2.7 ([MM96, Th. 7]). *For any set system $\mathcal{F}$, and $\emptyset \neq H \in \overline{\mathcal{F}} \cap \mathcal{F}^{\perp}$ we have the following.*

$$\texttt{valid}\,(U, \mathcal{F}) = \texttt{valid}\,(U/H, \mathcal{F}/H) * \texttt{valid}\left(H, \mathcal{F} \cap 2^{H}\right)$$

The idea behind Theorem 2.2.7 is as follows. A permutation α of U is a composition of two permutations with respect to H - (i) a permutation γ of H and (ii) a permutation β of U/H.

For two mutually orthogonal sets A, B such that $A \nsubseteq B$, A/B is defined as the set obtained by removing all elements of B from A and adding a representative element for B in A. Being orthogonal, A/B results in only the following three possibilities.

i. A and B are disjoint: $A/B = A$

ii. A is a subset of B: A/B is not defined

iii. B is a subset of A: $A/B = A \setminus B \cup \{x_B\}$, where x_B is a new element not in U added to represent B.

We observe that this idea of decomposing the COP problem into two subproblems in [MM96] is very similar to the substitution decomposition of a set system given in [McC04, Sec. 4]. [8]

The following Corollary 2.2.8 states how PQR-tree elegantly fits into this whole theory and help in computing all COP orders of $\mathcal{F}$, if any.

Corollary 2.2.8 ([MM96, Cor. 8]). *Let $\mathcal{F}$ be a set system with universe U and H is a non-empty orthogonal set $H \in \overline{\mathcal{F}} \cap \mathcal{F}^{\perp}$. If there is a PQR-tree T_1 that encodes all permutations in $\texttt{valid}\,(U/H, \mathcal{F}/H)$ and also a PQR-tree T_2 that encodes all permutations $\texttt{valid}\left(H, \mathcal{F} \cap 2^{H}\right)$, then a PQR-tree T for $\mathcal{F}$ can be obtained by replacing the leaf x_h in T_1 by T_2, where x_H is the representative element for H in T_i.*

Thus we have a recursive algorithm that can compute the PQR-tree for $\mathcal{F}$ provided we find an element from $\overline{\mathcal{F}} \cap \mathcal{F}^{\perp}$ in each iteration. The non-empty sets in $\overline{\mathcal{F}} \cap \mathcal{F}^{\perp}$ are called *node sets* since they form the nodes in the PQR-tree, as alluded to by Corollary 2.2.8. They are calculated as follows. First method is by computing the overlap components of $\mathcal{F}$. The overlap components is the partition that results from the *overlap* equivalence relation which is nothing but *non-orthogonal* equivalence relation (It is easy to verify

8. Using the theory cited in footnotes 5,6,7.

that this relation is indeed an equivalence relation.). Overlap components are linearly computable[MM95, Hsu92]. Once these elements are factored out, the rest of the node sets are obtained by the second method which is identifying *twin* elements [9]. Two elements $a, b \in U$ are twins if their membership in every set of $\mathcal{F}$ is in tandem with each other, i. e. $\{a, b\} \perp \mathcal{F}$. This is clearly an equivalence relation and their equivalence classes is known to be computable in linear time[Hsu01] and even in logspace[KKLV10].

The recursion end condition is when one cannot find any more sets from $\overline{\mathcal{F}} \cap \mathcal{F}^{\perp}$ that are non-trivial. This is when $\overline{\mathcal{F}} \cap \mathcal{F}^{\perp} = \mathcal{T}(U)$. This is the point where the parents of the leaves of the final PQR-tree are created. The following theorem helps the algorithm decide whether a P, Q or R node must be created.

Theorem 2.2.9 ([MM96, Th. 9]). *If $\mathcal{F}$ is a set system with universe U and $|U| \geq 3$ then one of the following statements hold.*

1. $\overline{\mathcal{F}} = \mathcal{T}(U)$
2. $\overline{\mathcal{F}} = consec(\alpha)$ *for some permutation α on U ($consec(\alpha)$ denotes a linear order)*
3. $\overline{\mathcal{F}} = 2^U$

In case (1) of Theorem 2.2.9 all elements in U are made children of a P node. In case (2) all elements in U are made children of a Q node in the order given by α. Finally, case (3) is the situation where no permutation of U gives COP. All elements of U are in this case made children in an R node.

Comparison with generalized PQ-tree

Stepping back a little, the reader may have noticed the cross references between PQR-tree and other data structures in footnotes earlier. Generalized PQ-tree or gPQ-tree is a data structure defined in [Nov89] to represent all orthogonal sets of a set system $\mathcal{F}$. A data structure with the same name was later defined in [McC04] as part of a *substitution decomposition* for a set system $\mathcal{F}$ and subsequently [McC04] gives a new characterization of $\mathcal{F}$ using a so-called incompatibility graph (this is discussed in more detail in Section 2.3.2). As seen above, PQR-tree is defined in [MM96] as a data structure to represent any set system $\mathcal{F}$ with additional information in their R-nodes if $\mathcal{F}$ has no COP. All three of these data structures are proposed as generalizations of [BL76]'s PQ-tree and hence produce the PQ-tree if $\mathcal{F}$ has COP. As a whole, all these three data structures are largely identical with their differences being notional. We discussed the basic theory that they all hold and predominantly used the terminology from [MM96]. To the best of our knowledge this comparison study has not been made earlier in the

9. [KKLV10, Sec. 3] calls this *indistinguishable* elements. The equivalence class is called a *slot*.

literature. We observe the three data structures of PQR-tree, gPQ-tree and generalized PQ-tree to be equivalent. The terminologies of the theory of all these data structures is summarised in Table 2.3.

Going back to the theory we were discussing earlier, Theorem 2.2.9 and the theory leading to it is very similar to [McC04, esp. Th. 2.1, 3.5. also Th. 3.2, 3.3, 3.4] which categorizes the nodes in the three cases of Theorem 2.2.9 as *prime*, *linear* and *degenerate* respectively. [McC04]'s generalized PQ-tree is created in similar ways as PQR-tree above. This tree is, in essence, the Hasse diagram of what they call *strong elements* of weak closure, $\mathcal{W}(\mathcal{F})$(i. e. $\overline{\mathcal{F}}$). Strong elements of a set family are elements that do not overlap with any other elements in the family, i. e. it is orthogonal to all other sets[McC04, Def. 3.3].

Theorem 2.2.10 ([MM96], [McC04, Th. 3.6]). *The set system $\mathcal{F}$ has COP if and only if its PQR-tree has no R nodes.*

Thus PQR-tree gives a data structure that encodes possible linear orderings of a set that demonstrates the COP property in it or narrows it down to parts of the universe in R nodes that prevent the set system from having COP.

We will now see, in brief, one more generalization of PQ-tree before seeing another approach to solving COP testing called ICPIA which is a set theoretic characterization of COP.

2.2.4 PC-tree– a generalization of PQ-tree

PC-tree is another generalization of PQ-tree. It is a data structure that is analogous to PQ-tree but for matrices with circular-ones property. PC-tree was introduced by [SH99] for the purpose of planarity testing where this data structure represents partial embeddings of planar graphs. In [Hsu01], Hsu reintroduces PC-tree as a generalization of PQ-tree and shows how it simplifies [BL76]'s planarity test by making the PQ-tree construction much less complicated. Later [HM03], discovers that PC-tree is a representation of all circular-ones property orders of a matrix when it is unrooted. PC-tree presented in [Hsu01] is rooted; however the construction of PC-tree is the same in both results. The property of being unrooted is necessary in order to use PC-tree as a data structure for encoding circular ordering. Definition 2.2.5 defines PC-tree.

Definition 2.2.5 (PC-tree [Hsu01, Dom08]). A PC-tree of matrix M with CROP on columns (rows), is a tree with the following properties.

i. Is unrooted – thus it has (a) no parent child relationship between nodes (b) there is no left to right (or vice versa) ordering.

gPQ-tree [Nov89]	PQR-tree [MM96]	generalized PQ-tree [McC04]
Trivial intersections	Trivial sets $\mathcal{T}(U)$	
Trivially intersecting sets	Mutually orthogonal sets, $A \perp B$	Non-overlapping sets
	Complete collection	Weakly partitive family
	$\overline{\mathcal{F}}$	Weak closure, $\mathcal{W}(\mathcal{F})$
	$\mathcal{F}^{\perp}$	Non-overlapping family, $\mathcal{N}(\mathcal{F})$
	Sets in $\mathcal{F}$ orthogonal to all other sets in $\mathcal{F}$	Strong elements
	PQR-tree	Decomposition tree of $\mathcal{W}(\mathcal{F})$, $T(\mathcal{W}(\mathcal{F}))$
	Node sets, $(\mathcal{F} \cap \mathcal{F}^{\perp}) \setminus \{\emptyset\}$	
	P-node $\Leftrightarrow \mathcal{F} = \mathcal{T}(\mathcal{F})$ Q-node $\Leftrightarrow \mathcal{F} = consec(\alpha)$ R-node $\Leftrightarrow \mathcal{F} = 2^U$	**Prime**: For every I, $\emptyset \neq I \subset [k], \cup_{i \in I} S_i \notin \mathcal{F}$ **Linear**: There exists an ordering of $[k]$ s.t. $\emptyset \neq I \subset [k]$ implies $\cup_{i \in I} S_i \in \mathcal{F}$ only if members of I are consecutive in the ordering **Degenerate**: For every I, $\emptyset \neq I \subset [k], \cup_{i \in I} S_i \in \mathcal{F}$

Table 2.3: Comparison of theory of [MM96] PQR-tree, [Nov89] gPQ-tree, [McC04] generalized PQ-tree

ii. Each leaf uniquely represents a row (column) of M. The leaf order of the tree gives a CROP order for column (row) for M. Moreover, any sequence obtained by considering the leaves in clockwise or counter-clockwise order describes a CROP order for M.

iii. Every non-leaf node in the tree is labeled P or C.

iv. The neighbors of P nodes can be permuted in any fashion to obtain a new CROP order for M.

v. The tree can be changed by applying the following "mirroring" operation to obtain a new CROP order for M. Root the PC-tree at a neighbor of a C-node, v and mirror the subtree whose root is v and finally unrooting the tree. Mirroring a subtree is done by putting the children of every node of the subtree in reverse order.

As a data structure when PC-tree is compared with PQ-tree, the differences are, (i) it is unrooted, (ii) it represents all CROP order of a matrix (iii) it has C nodes instead of Q nodes which can be "mirrored" (operation defined in Definition 2.2.5 v). The algorithms of construction of PQ-tree in [BL76] and that of PC-tree in [Hsu01, HM03] starkly differ since the latter is a much simplified procedure.

r_1	**1**	0	**1**	0	0	0	0	0
r_2	**1**	**1**	0	0	0	0	0	0
r_3	0	**1**	**1**	0	0	0	0	0
r_4	0	0	**1**	**1**	0	0	0	0
r_5	0	0	**1**	**1**	**1**	0	0	0
r_6	0	0	**1**	0	**1**	0	0	0
r_7	0	0	**1**	0	0	**1**	0	0
r_8	0	0	**1**	0	0	**1**	**1**	0
r_9	0	0	**1**	0	0	0	**1**	0
r_{10}	0	0	**1**	0	0	0	0	**1**

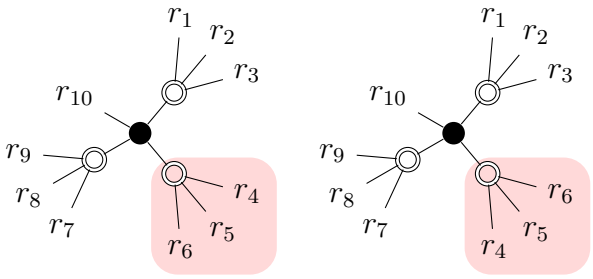


Figure 2.5: PC-tree with the associated matrix. The dark circular nodes are P-nodes and the double circular nodes are C-nodes. The two PC-trees are both for the same matrix. The second tree is obtained by performing the "mirroring" operation on the subtree (highlighted in the figure) rooted at the bottom right C-node. [Dom08, Fig. 2.11]

2.2.5 ICPIA - a set cardinality based COP test

In this section we see a characterization that does not use PQ-tree. [NS09] describes a characterization of consecutive-ones property solely based on the cardinality properties of the sets in the set system and does not use any variants of PQ-trees.

Definition 2.2.6 (Intersection Cardinality Preserving Interval Assignment (ICPIA)). Let $\mathcal{F} = \{A_i \mid A_i \subseteq U, i \in [m]\}$ be a set system from universe U and the set of ordered pairs $\Pi = \{(A_i, B_i) \mid B_i$ is an interval from $[n], i \in [m]\}$ be an *interval assignment* of $\mathcal{F}$, then Π is called an *ICPIA* if it has the following properties.

i. $|A_i| = |B_i|$ for all $i \in [m]$

ii. $|A_i \cap A_j| = |B_i \cap B_j|$ for all $i, j \in [m]$

Theorem 2.2.11 ([NS09, Th. 1]). *If an interval assignment Π is feasible, then it is an ICPIA.*

The necessity condition of Theorem 2.2.11 is fairly obvious. It turns out that it is sufficient too. This is demonstrated by two algorithms which work as follows. The first one iterates over the assignment Π and changes it in each iteration till the altered assignment has no more overlapping intervals (intervals are the second element in each ordered pair of Π. Def. 2.2.6) [NS09, Alg. 1]. This is achieved by replacing two overlapping assignment pairs (A_i, B_i) and (A_j, B_j) with the partitions of $(A_i \cup A_j, B_i \cup B_j)$ induced by their overlaps – $(A_i \setminus A_j, B_i \setminus B_j)$, $(A_i \cap A_j, B_i \cap B_j)$ and $(A_j \setminus A_i, B_j \setminus B_i)$.

To see it in terms of the theory explained in Section 2.2.3, this results in the set of strong elements of the weak closure of $\mathcal{F}$. Let $\mathcal{F}, \mathcal{I}$ be the set system and interval system (a set system where all the sets are intervals), respectively, involved in the assignment Π and $\mathcal{F}'$, $\mathcal{I}'$ be the same for the output of the algorithm Π'. It can be observed that $\mathcal{F}' \cup \mathcal{F}$ is $\overline{\mathcal{F}}$ or the weak closure of $\mathcal{F}$ without the set unions. The analog is true for $\mathcal{I} \cup \mathcal{I}'$.

Moreover, this output $\mathcal{F}'$, $\mathcal{I}'$ contains the strong elements of $\overline{\mathcal{F}}$, $\overline{\mathcal{I}}$ respectively. It turns out that $\mathcal{I}'$ is also an interval system and that Π' is an ICPIA [NS09, Lem. 2].

The next algorithm [NS09, Alg. 2] further refines Π' to Π'' which represents the family of permutations that yield the COP orders represented by Π of $\mathcal{F}$. The subset relation Hasse diagram of $\mathcal{I}'$ is created with each node representing the corresponding assignment pair of the interval from Π'. Since all intervals in Π' are non-overlapping, this is a tree. Notice that this means any two intervals are now either disjoint or one is contained in the other. Hence this tree is called a *containment tree*. The algorithm traverses this tree in post order fashion where the post order function is to replace an assignment pair (X, Y) in Π' by their child assignment pairs in the containment tree, say $(X_1, Y_1), (X_2, Y_2) \ldots (X_k, Y_k)$. The result Π'' gives the COP order for Π. These two algorithms prove that ICPIA is a sufficient condition for the feasibility of the interval assignment.

Theorem 2.2.12 ([NS09, Th. 2]). *Let $\mathcal{F} = \{A_i \mid A_i \subseteq U, i \in [m]\}$ be a set system from universe U, $|U| = n$ and $\Pi = \{(A_i, B_i) \mid B_i$ is an interval from $[n], i \in [m]\}$ be an ICPIA. Then there exists a permutation $\sigma : [n] \to [n]$ such that $\sigma(A_i) = B_i$.*

What we saw so far in this section is checking for the feasibility of a given interval assignment. ICPIA can also be used to compute an interval assignment when only the set system $\mathcal{F}$ is given. Part of the approach is similar to the overlap component idea in [FG65, Hsu02]. Once the overlap components are factored out, they can be independently assigned a subinterval to which the sets in the component need to be mapped. This is simple because the overlap components are disjoint from each other. Now the subproblem is finding COP order for each overlap component. In each component, the set whose overlaps with the rest of the sets in that component forms a single inclusion chain (see Definition 1.3.7; in this particular reference, the set of sets is the set of overlaps) is chosen first and assigned the leftmost interval. Chosing leftmost interval is only a convention – rightmost can also be chosen analogously to get a different COP order. The next candidate set is chosen from this set's overlapping sets. The interval assigned is calculated by simple intersection cardinality means. Assume that the next set overlaps on one of the "ends" of the current set and calculate the interval by shifting left or right of the current interval and adjusting its size to match the next set's cardinality. Once the next set is assigned an interval, the current assigned sets are all checked for ICPIA. This is a backtracking algorithm – if at any point ICPIA fails, another overlapping set is chosen. If all overlapping set attempts fail ICPIA, a different "first set" is chosen. This is a remarkably simple algorithm due to its obvious simplicity in implementation.

The work of this thesis described in Chapter 3 is largely an extension of the ICPIA characterization. In the new work, we adopt the notation used in [KKLV10] namely, that of hypergraph isomorphism (also described in that chapter).

2.3 Matrices without COP

COP is a very beneficial property since it simplifies the structure of the input (binary matrix or set system) leading to efficient algorithms to otherwise hard problems. While Section 2.2 discusses matrices with COP and how to compute COP orders, this section describes a couple of problems when the input does not have COP and identify their forbidden substructures that prevents them from having COP.

2.3.1 Finding forbidden submatrices

It is known from Theorem 2.2.4 that when a matrix does not have COP, it must have one of Tucker's forbidden submatrices (See Section 2.2.1 for detailed discussion on Tucker's forbidden substructures for matrices with COP). However, none of the COP test algorithms discussed in Section 2.2 outputs the forbidden submatrix when the test is negative. Using a reduction to the longest path problem, [Vel82] proves that finding a Tucker's forbidden submatrix of at least k rows is NP complete. This is a maximization problem. An example of an optimization problem for a matrix without COP is finding if deleting at most k rows from input will restore COP. This is known to be NP complete [Boo75]. There are several such optimization problems that try to find ways to get COP on the input matrix with certain alterations and finding smallest forbidden submatrix is at the heart of these problems. This minimization problem of finding smallest forbidden submatrix is polynomial time solvable.

In this section we will summarize [Dom08]'s algorithm for finding forbidden submatrix. We will use $\mathbb{T}$ to refer to the set of Tucker's forbidden submatrices shown in Figure 2.3.

A corollary to one of Tucker's theorems presented in Theorem 2.2.2 is given in Corollary 2.3.1. This is central to [Dom08]'s algorithm.

Corollary 2.3.1 ([Dom08, Cor. 3.1]). *A matrix M has COP if and only if its representing bipartite graph G_M (the graph whose half adjacency matrix is M) does not contain an asteroidal triple whose three vertices correspond to columns of M.*

Using Corollary 2.3.1, a forbidden submatrix can be found as follows. For every vertex triple x, y, z in G_M that correspond to columns of M, determine the sum of the lengths of three shortest paths connecting each pair of these vertices (there are 3 possible pairs) avoiding the neighborhood of third vertex. If all three paths exist, then x, y, z indeed form an asteroidal triple. Among all such triples, select the triple u, v, w where the sum is minimum and return the rows and columns of M that correspond to the vertices in the three shortest paths calculated earlier for this triple. This return value forms a submatrix and it obviously must contain a submatrix from $\mathbb{T}$ because it represents an

asteroidal triple. It may be noted that this procedure does not always return a submatrix of smallest size because the shortest paths computed may not be vertex disjoint. The following claim shows that this certainly gives a reasonable approximation of the solution. The claim has been abridged to summarize the main idea behind this algorithm. Note that a (x, y)-matrix is one with at most x number of **1**s in every column and at most y number of **1**s in every row. If x or y is given as $*$, it denotes that number of **1**s is unbounded.

Proposition 2.3.2 ***([Dom08, Pr. 3.1,* abridged*]).*** Let M be a $(*, \Delta)$-matrix of size $m \times n$ that contains a forbidden submatrix M' of size $m' \times n'$ from $\mathbb{T}$. Then this algorithm computes a submatrix that belongs to $\mathbb{T}$ with at most $m' + 5$ rows and $n' + 3$ columns in $O(\Delta mn^2 + n^3)$ time.

In fact, the approximation mentioned in Proposition 2.3.2 is the worst case which happens to be when the actual solution M' is of type M_{IV}. The approximation is slightly better for the other types of submatrices from $\mathbb{T}$, the best case being an exact solution when M' is M_{I_k} or M_{II_k}. Thus this algorithm works accurately when the solution is of type M_{I_k} or M_{II_k}.

Next, notice that Tucker matrix types M_{IV} and M_V are constant size – 4×6 and 4×5 respectively. Thus they can be searched for in polynomial time even with brute force method as Corollary 2.3.3 states.

Corollary 2.3.3 ([Dom08, Cor. 3.3, *abridged*]). *Let M be a binary $(*, \Delta)$-matrix of size $m \times n$ and M' be a binary matrix from $\mathbb{T}$. A submatrix in M that is isomorphic to M' can be found in $O(\Delta^3 m^2 n^3)$ time or $O(\Delta^4 m^2 n)$ time when M' is of type M_{IV} or M_V respectively.*

Their algorithm for finding M_{III_k} type submatrix is slightly involved and we only present here the analysis result in Proposition 2.3.4. The algorithm is derived from the structural similarities between the first three types of Tucker matrices and the fact that M_{I_k} is a chordless cycle of length $2k + 4$.

Proposition 2.3.4 ***([Dom08, Pr. 3.6]).*** Let M be a $(*, \Delta)$-matrix M of size $m \times n$. Then a minimum size submatrix of type M_{III_k} in M can be found in $O(\Delta^3 m^3 n + \Delta^2 m^2 n^2)$ time.

All the above algorithmic results are combined to obtain the final single algorithm for finding forbidden submatrix. Theorem 2.3.5 gives this result.

Theorem 2.3.5 ([Dom08, Th. 3.1]). *Let M be a $(*, \Delta)$-matrix of size $m \times n$. A forbidden submatrix from $\mathbb{T}$ in M that has a minimum number of rows can be found in $O(\Delta^3 m^2 n(m + n^2))$ time. Within the same time, one can also find a forbidden submatrix from $\mathbb{T}$ in M that has a minimum number of columns, a minimum number of rows and*

columns, or a minimum number of ***1****entries.*

The procedure used to prove Theorem 2.3.5 is as follows.

1. Run the algorithm mentioned in Proposition 2.3.2 to obtain a submatrix A which is an approximation of the smallest forbidden submatrix. (If the actual solution is M_{I_k} or M_{II_k} this will be an exact solution.)
2. Run the algorithm from Proposition 2.3.4 to find an induced M_{III_k} and let this be B.
3. Run the algorithm from Corollary 2.3.3 twice – once each for finding M_{IV} and finding M_V. Let the results be C and D respectively.
4. Return the minimum size matrix among A, B, C, D (minimum size being one of the following: number of rows, number of columns, sum of number of rows and columns, number of **1** entries).

Note that this algorithm has a factor Δ in the time complexity analysis, Δ being the maximum number of **1**s in a row. For sparse matrices this is much lesser than the total number of columns m. When the matrix is not sparse, the worst case is when Δ is almost equal to m (if they are equal quantities, that row can be disregarded from input matrix since a row with all **1**s is trivially consecutive). Thus this is a polynomial time algorithm although a more efficient term in the running efficiency is desirable.

In the next section we see yet another forbidden substructure for COP which is can be computed in linear time.

2.3.2 Incompatibility graph - a certificate for no COP

[McC04] describes another characterization for matrices without COP which is linearly computable. This is done by way of an *incompatibility graph* of a set system (since COP in matrices and set systems are equivalent problems as seen in Section 2.2.3 we will use the term set system in this section, in line with [McC04]'s convention). The construction of this graph is based on the following observation. A similar observation was made in [COR98] in relation to the betweenness problem in computational biology.

Observation 2.3.1. Consider elements $a, b, c \in U$ and a set system $\mathcal{F}$ with universe U. If there is at least one set $S \in \mathcal{F}$ such that $a, c \in S$ but $b \notin S$, then it is impossible to have a COP order that will place b in between a and c.

For the sake of argument, suppose there exists such a COP order. Clearly, it will not succeed in mapping S to an interval since $b \notin S$ which is a contradiction to the

assumption that this is a COP order. Let the ordered pair (x, y) denote the condition that in the COP order y comes after x. Based on above observations, this means (a, b) and (b, c) are not compatible in any COP order. So are (c, b) and (b, a). Thus a binary relation *is incompatible with* on $U \times U \setminus \{(a, a) \mid a \in U\}$ can be defined based on the membership of elements of U in the sets in $\mathcal{F}$. [McC04] creates an incompatibility graph with vertices denoting ordered pairs and edges denoting the incompatibility relation.

Definition 2.3.1 (Incompatibility graph [McC04, Def. 6.1]). Let $\mathcal{F}$ be a set system with universe U. Let $A_U = \{(a, b) \mid a, b \in U, a \neq b\}$. The *incompatibility graph* $G_I^{\mathcal{F}}$ of $\mathcal{F}$ is an undirected graph defined as follows.

1. The vertex set of $G_I^{\mathcal{F}}$ is A_U
2. The edge set of $G_I^{\mathcal{F}}$ are pairs of the following forms
   - $\{(a, b), (b, a)\}$ for all $a, b \in U$
   - $\{(a, b), (b, c)\}$ if there exists $S \in \mathcal{F}$ such that $a, c \in S$ and $b \notin S$, for all $a, b, c \in U$

Thus an edge between two vertices $(a, b), (b, c)$ in $G_I^{\mathcal{F}}$ means that there exists no linear order on U that can place b after a and place c after b. For any COP order, there must not be any incompatible pairs. Thus it must consist of an independent set I of the incompatibility graph. Moreover the independent set must have exactly half the vertex set of $G_I^{\mathcal{F}}$ since the COP order must involve all elements in U and each vertex denotes a pair of elements. Secondly, the reverse of a COP order is also a COP order. Thus $A_U \setminus I$ must also be an independent set. In other words, $G_I^{\mathcal{F}}$ must be bipartite with partitions being of equal size if $\mathcal{F}$ has COP. An odd cycle is a forbidden structure in a bipartite graph, hence the same is forbidden in the incompatibility graph of a set system with COP. In other words, if the set system has no COP, its incompatibility graph must have an odd cycle. There is also a requirement on the length of this odd cycle which is formally stated in Theorem 2.3.6.

Theorem 2.3.6 ([McC04, Th. 6.1], [Dom08, Corrected by Th. 2.6, Proof p. 44–47]). *Let $\mathcal{F}$ be a set family with universe U. Then $\mathcal{F}$ has COP if and only if its incompatibility graph is bipartite and if it does not have the COP, the incompatibility graph has an odd cycle of length at most $n + 3$.*

Searching for an odd cycle to recognize a bipartite graph is a known and polynomially solved problem. [McC04]'s algorithm also labels each edge of the cycle with a set from $\mathcal{F}$ that documents the incompatibility.

CHAPTER 3

TREE PATH LABELING OF PATH HYPERGRAPHS – NEW RESULTS

This chapter documents all the new results obtained by us in the area of tree path labeling of path hypergraphs which is the problem addressed in this thesis. The organization of the chapter is as follows.

Section 3.1 recalls the idea of tree path labeling. The necessary preliminaries with definitions etc. are presented in Section 3.2. Section 3.3 documents a characterization for feasible path labelings. In Section 3.4 we demonstrate the algorithm described in Section 3.3 by working out the solution to an example. Section 3.5 describes two special cases where the target tree is of a particular family of trees. The first one is shown to be equivalent to COP testing in Section 3.5.1. Section 3.5.2 discusses the second special case and presents a polynomial time algorithm to find the tree path labeling of a given set system from a given k-subdivided star. Section 3.6 discusses the plain vanilla version where the target tree has no restrictions and the algorithm to find a feasible TPL, if any, in this case. Finally, in Section 3.7 we discuss the complexity of TPL and we conclude by showing that consecutive-ones property testing, which is a special case of TPL, is in LOGSPACE.

3.1 Summary of Proposed Problems

In Section 1.5 we see that consecutive-ones property and its equivalent problem of interval labeling of a hypergraph is a special case of the general problem of tree path labeling of

path hypergraphs. Moreover, Section 2.2.5 describes how the problem of consecutive-ones property testing can be easily seen as a simple constraint satisfaction problem involving a hypergraph or a system of sets from a universe. Every column (row) of the binary matrix can be converted into a set of non-negative integers which are the indices of the rows (columns) with **1**s in that column (row). When observed in this context, if the matrix has the COP on columns (rows), a reordering of its rows (columns) will result in sets that have only consecutive integers. In other words, the sets after applying the COP row (column) permutation are intervals. In this form, one can see that this is indeed the problem of finding interval assignments to the given set system with a single permutation of the universe (set of row or column indices for COP of columns or rows, respectively) which permutes each set to an interval. The result in [NS09] (see Theorems 2.2.11 and 2.2.12) characterizes interval assignments to the sets which can be obtained from a single permutation of the universe. The cardinality of the interval assigned to it must be same as the cardinality of the set, and the intersection cardinality of any two sets must be same as the intersection cardinality of the corresponding intervals. This is a necessary and sufficient condition.

Intervals are paths from a tree with maximum degree two. Thus the interval assignment problem can be naturally generalized into path assignment problem from any tree. We refer to this as the *tree path labeling problem of path hypergraphs*. This is analogous to the interval labeling problem in literature [KKLV10] to interval hypergraphs. In brief, the problem is defined as follows – given a hypergraph $\mathcal{F}$ from universe U (i. e. hypergraph vertex set) and a target tree T, does there exist a bijection ϕ from U to the vertices of T such that for each hyperedge when ϕ is applied to its elements it gives a path on T. More formally, the problem definition is given by Compute Feasible Tree Path Labeling.

Compute Feasible Tree Path Labeling

`Input`	A hypergraph $\mathcal{F}$ with vertex set U and a tree T.
`Question`	Does there exist a set of paths $\mathcal{P}$ from T and a bijection $\ell : \mathcal{F} \to \mathcal{P}$, such that Feasible Tree Path Labeling returns **true** on $(\mathcal{F}, T, \ell)$.

To characterize a feasible TPL, we consider the case where a tree path labeling is also given as input and we are required to test if the given labeling is feasible. This is defined by the Feasible Tree Path Labeling problem.

FEASIBLE TREE PATH LABELING

Input	A hypergraph $\mathcal{F}$ with vertex set U, a tree T, a set of paths $\mathcal{P}$ from T and a bijection $\ell : \mathcal{F} \to \mathcal{P}$.
Question	Does there exist a bijection $\phi : U \to V(T)$ such that ϕ when applied on any hyperedge in $\mathcal{F}$ will give the path mapped to it by the given tree path labeling ℓ. i.e., $\ell(S) = \{\phi(x) \mid x \in S\}$, for every hyperedge $S \in \mathcal{F}$.

Section 3.3 discusses FEASIBLE TREE PATH LABELING and presents an algorithmic characterization for a feasible TPL.

With respect to computing a feasible TPL, as suggested by COMPUTE FEASIBLE TREE PATH LABELING problem, we were unable to discover an efficient algorithm for it. However, we consider two special cases of the same on special target trees – namely, COMPUTE INTERVAL LABELING and COMPUTE k-SUBDIVIDED STAR PATH LABELING on intervals and k-subdivided stars (see Definition 3.2.6), respectively. Section 3.5 discusses these problems and their polynomial time solutions. COMPUTE INTERVAL LABELING is the known problem of COP testing. The polynomial time solution to COMPUTE k-SUBDIVIDED STAR PATH LABELING is a new result by us.

COMPUTE INTERVAL LABELING

Input	A hypergraph $\mathcal{F}$ with vertex set U and a tree T with maximum degree 2.
Question	Does there exist a set of paths $\mathcal{P}$ from T and a bijection $\ell : \mathcal{F} \to \mathcal{P}$, such that FEASIBLE TREE PATH LABELING returns **true** on $(\mathcal{F}, T, \ell)$.

COMPUTE k-SUBDIVIDED STAR PATH LABELING

Input	A hypergraph $\mathcal{F}$ with vertex set U such that every hyperedge $S \in \mathcal{F}$ is of cardinality at most $k+2$ and a k-subdivided star T.
Question	Does there exist a set of paths $\mathcal{P}$ from T and a bijection $\ell : \mathcal{F} \to \mathcal{P}$, such that FEASIBLE TREE PATH LABELING returns **true** on $(\mathcal{F}, T, \ell)$.

An algorithm for COMPUTE FEASIBLE TREE PATH LABELING on arbitrary trees is presented in Section 3.6 which would run in polynomial time if two subproblems we identify are polynomial time solvable.

3.2 Preliminaries to new results

This section describes ideas and terminologies necessary in this chapter. Some of these may have been seen earlier in this document, but is recalled for the sake of completeness of the chapter.

A note on attribution – the machinery for labeling of hypergraphs has been adopted from [KKLV10] and adapted to the needs of our work. Some classic graph definitions are from [Gol04].

3.2.1 Set systems and Hypergraphs

Consider a universe U with $|U| = n$. The set $\mathcal{F} \subseteq (2^U \setminus \emptyset)$ is a *set system* with universe U. The *support of set system* $\mathcal{F}$ denoted by $supp(\mathcal{F})$, is the union of all the sets in $\mathcal{F}$.

$$supp(\mathcal{F}) = \bigcup_{S \in \mathcal{F}} S$$

By convention, the set system $\mathcal{F}$ "covers" its universe: $supp(\mathcal{F}) = U$. If the sets in $\mathcal{F}$ are intervals or paths of a tree, then it is called an interval system or a path system, respectively.

A set system $\mathcal{F}$ can also be visualized as a *hypergraph* $\mathcal{F}_H$ whose vertex set is U and hyperedges are the sets in $\mathcal{F}$. This is a known representation for interval systems in literature [BLS99, KKLV10]. We extend this definition here to path systems. Due to the equivalence of set system and hypergraph as far as this thesis is concerned, we drop the subscript $_H$ in the notation and refer to both the structures by $\mathcal{F}$. Thus hypergraphs and set systems are synonyms. So are hyperedges and sets.

The notion of *hypergraph isomorphism* is central to the work of this thesis. This is defined by Definition 3.2.1.

Definition 3.2.1 (Hypergraph isomorphism). Let $\mathcal{F}'$, $\mathcal{F}''$ be two hypergraphs. They are said to be *isomorphic* to each other, denoted by $\mathcal{F}' \cong \mathcal{F}''$, if and only if there exists a bijection $\phi : supp(\mathcal{F}') \to supp(\mathcal{F}'')$ such that for all sets $A \subseteq supp(\mathcal{F}')$, A is a hyperedge in $\mathcal{F}'$ if and only if B is a hyperedge in $\mathcal{F}''$ where $B = \{\phi(x) \mid x \in A\}$ (this is also written as $B = \phi(A)$).

Figure 3.1 shows an example of hypergraphs and hypergraph isomorphism.

Recall intersection graph and path graph from Definition 1.3.9 and 1.3.10. The *intersection graph* $\mathbb{I}(\mathcal{F})$ of a hypergraph or set system $\mathcal{F}$ is a graph such that its vertex set has a bijection to $\mathcal{F}$ and there exists an edge between two vertices if and only if their corresponding hyperedges have a non-empty intersection. A graph G is a path graph if it is isomorphic to the intersection graph of a path system.

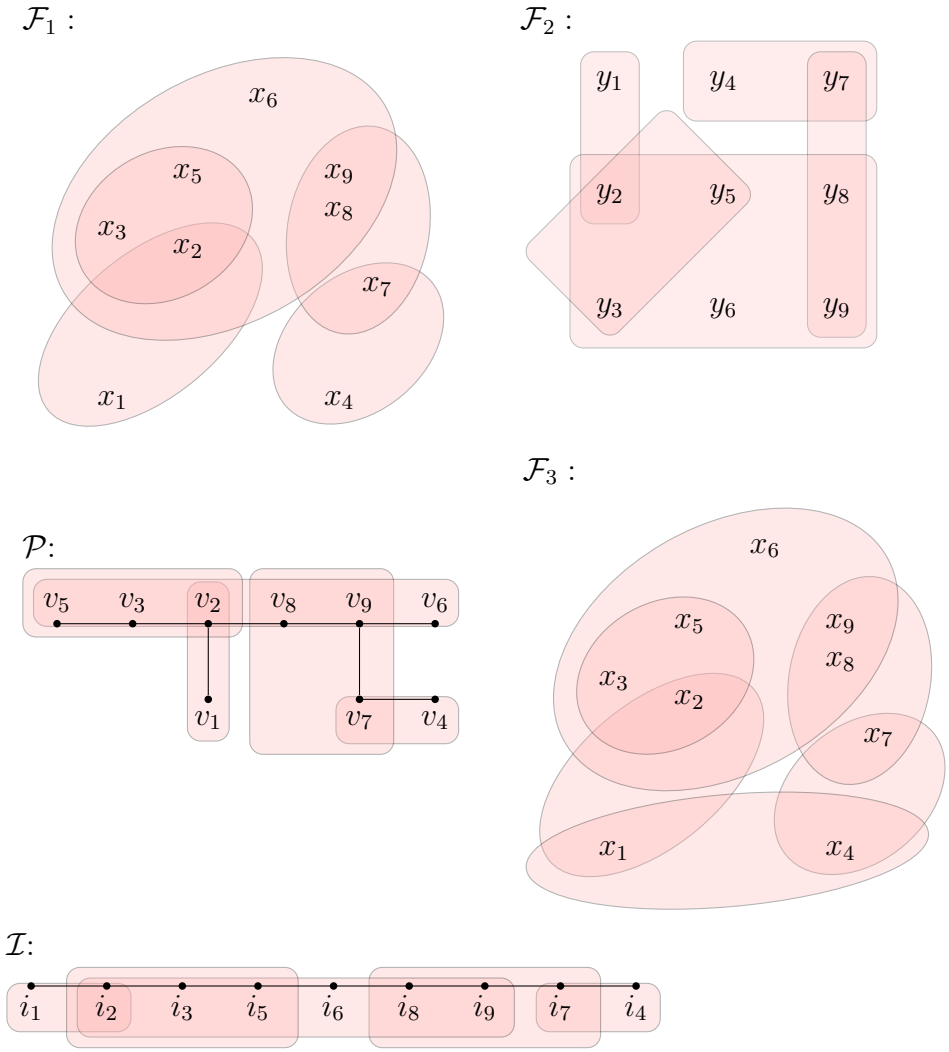


Figure 3.1: Hypergraphs and set systems. $\mathcal{F}_1$ and $\mathcal{F}_2$ are isomorphic hypergraphs. The subscript of each element is such that if ϕ is the hypergraph isomorphism, $\phi(x_i) = y_i$. $\mathcal{I}$ and $\mathcal{P}$ are an interval system and a path system respectively which are both isomorphic to $\mathcal{F}_1$ (and by transitivity, isomorphic to $\mathcal{F}_2$). Here too the subscript of elements indicate the isomorphism bijection. $\mathcal{F}_3$ is a hypergraph that is not isomorphic to any of the other hypergraphs – moreover, it is one that cannot have a feasible path labeling to any tree since it does not have the Helly's property [Gol04, Pr. 4.7] anymore due to the addition of hyperedge $\{x_1, x_4\}$.

3.2.2 Path Labeling and Path Hypergraphs

The graph T represents a *target tree* with same number of vertices as elements in U; $|V(T)| = |U| = n$. A *path system* $\mathcal{P}$ is a set system of paths from T.

$$\mathcal{P} \subseteq \{P \mid P \subseteq V,\ T[P] \text{ is a path}\}$$

A special mapping of paths from target tree T to hyperedges in $\mathcal{F}$ is called a *path labeling*. Definition 3.2.2 gives details. Note that the term "tree path labeling" is synonymous with "path labeling" in this document since we only consider trees and not general graphs as source of paths.

Definition 3.2.2 (Path labeling of a hypergraph). Consider a hypergraph $\mathcal{F}$ and a path

system $\mathcal{P}$ from tree T such that $\mathbb{I}(\mathcal{F}) \cong \mathbb{I}(\mathcal{P})$. Then the associated bijection $\ell : \mathcal{F} \to \mathcal{P}$ due to this isomorphism is called a **path labeling** of $\mathcal{F}$ from T. The path system $\mathcal{P}$ is alternatively denoted as $\mathcal{F}^{\ell}$. The path labeling may also be denoted by ordered pairs $(\mathcal{F}, \ell)$ or $(\mathcal{F}, \ell, T)$.

To elaborate on the phrase "associated bijection" in Definition 3.2.2, let $g : V(\mathbb{I}(\mathcal{F})) \to V(\mathbb{I}(\mathcal{P}))$ be the given intersection graph isomorphism, where $V(\mathbb{I}(\mathcal{F})) = \{v_S \mid S \in \mathcal{F}\}$ and $V(\mathbb{I}(\mathcal{P})) = \{v_P \mid P \in \mathcal{P}\}$. Then the associated bijection $\ell : \mathcal{F} \to \mathcal{P}$ is defined as follows. For all $S' \in \mathcal{F}$ and $P' \in \mathcal{P}$, $\ell(S') = P'$ if and only if $g(v_{S'}) = v_{P'}$. Thus, it can be seen that a path labeling preserves the non-empty intersection property between every pair of hyperedges in $\mathcal{F}$.

At this point, it is worth comparing Definition 3.2.2 with Definition 3.2.1 seen in the previous section. It may be noted that there are two kinds of isomorphisms here. One is the isomorphism of intersection graphs of $\mathcal{F}$ and $\mathcal{P}$, i.e. $\mathbb{I}(\mathcal{F})$ and $\mathbb{I}(\mathcal{P})$ and second is the isomorphism between the hypergraphs $\mathcal{F}$ and $\mathcal{P}$ themselves. It is easy to verify that given a path labeling of hypergraph $\mathcal{F}$, there need not be a hypergraph isomorphism involved. In fact, checking if an induced hypergraph isomorphism results from a given path labeling is the problem of Compute Feasible Tree Path Labeling. However, a hypergraph isomorphism always induces a path labeling as shown by the following Definition 3.2.3.

Definition 3.2.3 (Hypergraph isomorphism induced path labeling). Let $\mathcal{F}$ be a hypergraph and $\mathcal{P}$ be a path system such that $\mathcal{F} \cong \mathcal{P}$. Let $\phi : supp(\mathcal{F}) \to supp(\mathcal{P})$ be this hypergraph isomorphism function. Then there is an **induced path labeling** denoted by $p_\phi : \mathcal{F} \to \mathcal{P}$ of the hypergraph such that $p_\phi(S) = \phi(S)$ (see Definition 3.2.1) for all $S \in \mathcal{F}$.

Combining the ideas of path labeling, hypergraph isomorphism and its induced path labeling we now define *feasibility* of a path labeling in Definition 3.2.4.

Definition 3.2.4 (Feasible path labeling). Let $\mathcal{F}$ be a hypergraph and $(\mathcal{F}, \ell)$ be a path labeling. $(\mathcal{F}, \ell)$ is called a **feasible path labeling** if the following conditions hold.

- $\mathcal{F} \cong \mathcal{F}^{\ell}$
- if the above hypergraph isomorphism is $\phi : supp(\mathcal{F}) \to supp(\mathcal{F}^{\ell})$, then $p_\phi = \ell$

This isomorphism ϕ is called the **feasibility hypergraph isomorphism** or feasibility isomorphism.

One of the problems solved in this thesis Feasible Tree Path Labeling characterizes feasible tree path labeling.

Now we will see the definition of a *path hypergraph* which is a simple idea extended from interval hypergraphs in [KKLV10].

Definition 3.2.5 (Path hypergraph). Let $\mathcal{F}$ be a set system. $\mathcal{F}$ is called a **path hypergraph** if there exists a target tree T and a path system $\mathcal{P}$ from T such that $\mathcal{F} \cong \mathcal{P}$. If $\mathcal{F}$ is thus a path hypergraph, $\mathcal{P}$ is called **path representation** of $\mathcal{F}$.

In other words, if there exists a feasible tree path labeling for a given hypergraph, it is a path hypergraph.

3.2.3 Overlap Graphs and Marginal Hyperedges

Recall the notion of *overlap* from Definition 1.3.5. Two hyperedges S and S' are said to overlap, denoted by $S \between S'$, if they have a non-empty intersection and neither is contained in the other.

An *overlap graph* $\mathbb{O}(\mathcal{F})$ of a hypergraph $\mathcal{F}$ is a graph such that its vertex set has a bijection to $\mathcal{F}$ and there exists an edge between two of its vertices if and only if their corresponding hyperedges overlap. Thus $\mathbb{O}(\mathcal{F})$ is a spanning subgraph of $\mathbb{I}(\mathcal{F})$ and not necessarily connected. Each connected component of $\mathbb{O}(\mathcal{F})$ is called an *overlap component*.

A hyperedge $S \in \mathcal{F}$ is called *marginal* if for all $S' \between S$, the overlaps $S \cap S'$ form a single inclusion chain [KKLV10] (see Definition1.3.7). Additionally, if S is such that it is contained in no other marginal hyperedge in $\mathcal{F}$, then it is called *super-marginal*.

3.2.4 Miscellaneous

A *star* graph is a complete bipartite graph $K_{1,p}$ (which is clearly a tree with p leaves). The vertex with maximum degree in a star is called its *center* and the edges are called *rays*. This thesis is interested in a star-like graph which is defined in Definition 3.2.6

Definition 3.2.6 (k-subdivided star). A **k-subdivided star** is a star with all its rays subdivided exactly k times. A **ray** of a k-subdivided star is the path from the center vertex to a leaf. It is clear that all rays of a k-subdivided star are of length $k + 2$.

Definition 3.2.7 (In-tree). An **in-tree** is a directed rooted tree in which all edges are directed toward to the root.

In-trees are used in Section 3.6 which describes our solution to Compute Feasible Tree Path Labeling problem.

Finally, note that as a convention in this thesis, the set I represents the index set $[m]$ or $[n]$. If index i is used without further qualification, it is intended that $i \in I$.

3.3 Characterization of Feasible Tree Path Labeling

In this section we discuss the FEASIBLE TREE PATH LABELING problem introduced in Section 3.1 and give a set cardinality based characterization for it.

For this characterization, we generalize the notion of the ICPIA [NS09]. We show that for a given hypergraph $\mathcal{F}$ with universe U, a target tree T, and path labeling from T to $\mathcal{F}$, there is a feasible bijection between U and $V(T)$ if and only if all intersection cardinalities among any three sets (not necessarily distinct) is same as the intersection cardinality of the paths assigned to them. This characterization is proved constructively and the algorithm outputs a feasible bijection between U and $V(T)$.

We now introduce a special kind of path labeling called *Intersection Cardinality Preserving Path Labeling (ICPPL)* in Definition 3.3.1.

Definition 3.3.1. Consider a path labeling $(\mathcal{F}, \ell)$ on the given tree T. We call $(\mathcal{F}, \ell)$ an **Intersection Cardinality Preserving Path Labeling (ICPPL)** if it has the following properties.

i. $|S| = |\ell(S)|$ for all $S \in \mathcal{F}$

ii. $|S_1 \cap S_2| = |\ell(S_1) \cap \ell(S_2)|$ for all distinct $S_1, S_2 \in \mathcal{F}$

iii. $|S_1 \cap S_2 \cap S_3| = |\ell(S_1) \cap \ell(S_2) \cap \ell(S_3)|$ for all distinct $S_1, S_2, S_3 \in \mathcal{F}$

In the remaining part of this section we show that $(\mathcal{F}, \ell)$ is feasible if and only if it is an ICPPL. Algorithm 3 takes $(\mathcal{F}, \ell)$ as input and computes the feasibility bijection if it is feasible or else reports failure. This algorithm recursively does two levels of filtering of $(\mathcal{F}, \ell)$ to make it simpler in terms of set intersections while retaining a non-empty subset of isomorphisms, if any, between $\mathcal{F}$ and $\mathcal{F}^{\ell}$. These two filters are `filter common leaf` and `filter fix leaf` which in summary are as follows:

1. `filter common leaf` Refine $\mathcal{F}$ such that the resulting labeling will not have paths that share a leaf thus each leaf being unique to a path. This is done by breaking the paths into subpaths and their corresponding preimage sets as described in Algorithm 1.

2. `filter fix leaf` Find the element in universe U that maps to each leaf in T as described in Algorithm 2.

Remove the leaves from T and their corresponding preimages from U and call the filters again. This is repeated until the resulting truncated tree is a path. The remaining mapping can be found using ICPIA.

First, we present Algorithm 1 which describes `filter common leaf`, and prove

its correctness. This algorithm refines the path labeling by processing pairs of paths in $\mathcal{F}^\ell$ that share a leaf until no two paths in the new path labeling share any leaf.

Algorithm 1 Refine ICPPL `filter common leaf` $(\mathcal{F}, \ell, T)$

$\mathcal{F}_0 \leftarrow \mathcal{F}$, $\ell_0(S) \leftarrow \ell(S)$ for all $S \in \mathcal{F}_0$
$j \leftarrow 1$
while there is $S_1, S_2 \in \mathcal{F}_{j-1}$ such that $\ell_{j-1}(S_1)$ and $\ell_{j-1}(S_2)$ have a common leaf in T **do**
  $\mathcal{F}_j \leftarrow (\mathcal{F}_{j-1} \setminus \{S_1, S_2\}) \cup \{S_1 \cap S_2, S_1 \setminus S_2, S_2 \setminus S_1\}$
    `Remove` S_1, S_2 `and add the ``filtered'' sets`
  for every $S \in \mathcal{F}_{j-1}$ s.t. $S \neq S_1$ and $S \neq S_2$ **do** $\ell_j(S) \leftarrow \ell_{j-1}(S)$ **end for**
  $\ell_j(S_1 \cap S_2) \leftarrow \ell_{j-1}(S_1) \cap \ell_{j-1}(S_2)$
    `Carry forward the path labeling for all existing sets other than` S_1, S_2
  $\ell_j(S_1 \setminus S_2) \leftarrow \ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2)$
    `Define path labeling for new sets`
  $\ell_j(S_2 \setminus S_1) \leftarrow \ell_{j-1}(S_2) \setminus \ell_{j-1}(S_1)$
  $j \leftarrow j + 1$
end while
$\mathcal{F}' \leftarrow \mathcal{F}_j$, $\ell' \leftarrow \ell_j$
return $(\mathcal{F}', \ell')$

In order to prove the correctness of Algorithm 1, we present a few lemma which are useful in subsequent arguments.

The first two lemma below are from results in [NS09]. They have been reworded to use the terminology adopted in this thesis. Lemma 3.3.1 shows that the preservation of pairwise intersection cardinality in an interval assignment is sufficient to preserve three way intersection cardinality. Lemma 3.3.2 shows that if a path labeling (i) has target tree as a path i. e. it is an interval labeling, and (ii) it preserves pairwise intersection cardinalities i. e. it is an ICPIA, then it is a feasible path labeling.

Lemma 3.3.1 ([NS09, Lem. 1]). *Let P be a path, S_1, S_2, S_3 be 3 sets, and T_1, T_2, T_3 be paths from P, such that $|S_i \cap S_j| = |T_i \cap T_j|, 1 \leq i, j \leq 3$. Then, $|S_1 \cap S_2 \cap S_3| = |T_1 \cap T_2 \cap T_3|$.*

Lemma 3.3.2 ([NS09, Th. 2]). *A path labeling $(\mathcal{F}, \ell)$ with target tree T as a path is feasible if and only if it is an ICPIA.*

Next, we see Lemma 3.3.3 which is a direct consequence of Definition 3.3.1 and is analogous to [NS09, Cor. 1].

Lemma 3.3.3. *If ℓ is an ICPPL, and $S_1, S_2, S_3 \in \mathcal{F}$, then $|S_1 \cap (S_2 \setminus S_3)| = |\ell(S_1) \cap (\ell(S_2) \setminus \ell(S_3))|$.*

Proof. Let $P_i = \ell(S_i)$, for all $1 \leq i \leq 3$. $|S_1 \cap (S_2 \setminus S_3)| = |(S_1 \cap S_2) \setminus S_3| = |S_1 \cap S_2| - |S_1 \cap S_2 \cap S_3|$. Due to properties (ii) and (iii) of ICPPL, $|S_1 \cap S_2| - |S_1 \cap S_2 \cap S_3| =$

$|P_1 \cap P_2| - |P_1 \cap P_2 \cap P_3| = |(P_1 \cap P_2) \setminus P_3| = |P_1 \cap (P_2 \setminus P_3)|$. □

For interval labeling, [NS09] shows that if it is an ICPIA i. e. pairwise intersection cardinalities are preserved, then d-way intersection cardinalities are preserved for $2 < d < n$. Thus, in a way, pairwise intersections "capture" feasibility of an interval labeling. In the same vein, the following lemma indicates that three-way intersection cardinality "captures" d-way intersection cardinalities for $3 < d < n$ in a tree path labeling.

Lemma 3.3.4. *Let $(\mathcal{F}, \ell)$ be an ICPPL and $\ell(S_i) = P_i$, $S_i \in \mathcal{F}$, $1 \leq i \leq 4$. Then, $|\cap_{i=1}^{4} S_i| = |\cap_{i=1}^{4} P_i|$.*

Proof. Consider the set of set intersections with S_1 $\mathcal{F}' = \{S_2 \cap S_1, S_3 \cap S_1, S_4 \cap S_1\}$ and let ℓ' be a tree path labeling of $\mathcal{F}'$ such that $\ell'(S_i \cap S_1) = P_i \cap P_1, 2 \leq i \leq 4$. Clearly, $P_2 \cap P_1$, $P_3 \cap P_1$, and $P_4 \cap P_1$ are subpaths of path P_1, thus equivalent to intervals. Due to the three way intersection cardinality preservation property of the ICPPL $(\mathcal{F}, \ell)$, this new tree path labeling $(\mathcal{F}', \ell')$ preserves pairwise intersection cardinalities. Now by applying Lemma 3.3.1 to the sets in $\mathcal{F}'$ and their corresponding path images due to ℓ', it follows that $|\cap_{i=1}^{4} S_i| = |\cap_{i=1}^{4} P_i|$. □

Corollary 3.3.5. *Let $(\mathcal{F}, \ell)$ be an ICPPL and $\ell(S_i) = P_i$, $S_i \in \mathcal{F}$, $1 \leq i \leq 4$. If P_i, $1 \leq i \leq 4$ are such that $P_1 \cap P_2$, $P_1 \setminus P_2$, $P_2 \setminus P_1$ are also paths, then:*

$$|(S_1 \setminus S_2) \cap S_3 \cap S_4| = |(P_1 \setminus P_2) \cap P_3 \cap P_4|$$
$$|(S_2 \setminus S_1) \cap S_3 \cap S_4| = |(P_2 \setminus P_1) \cap P_3 \cap P_4|$$

Proof. It is clear that $|(P_1 \setminus P_2) \cap P_3 \cap P_4| = |\cap_{i=1}^{4} P_i| - |P_2 \cap P_3 \cap P_4|$. Since we have an ICPPL, from Lemma 3.3.4, it follows that the intersection cardinalities are preserved. □

Lemma 3.3.6. *Let $(\mathcal{F}, \ell)$ be an ICPPL and $\ell(S) = P$ for some $S \in \mathcal{F}$. If sets S_{priv} and P_{priv} contain only those elements of S and P that occur in no set other than S and P, respectively, then $|S_{priv}| = |P_{priv}|$.*

Proof. Consider the set system $\mathcal{F}' = \{S_i \cap S \mid S_i \in \mathcal{F}, i \in [n]\}$ and path labeling $\ell'(S_i \cap S) = P_i \cap P$. Since $(\mathcal{F}, \ell)$ is an ICPPL, it can be easily verified that $(\mathcal{F}', \ell')$ is an ICPIA on path (interval) P. Let S_{two} and P_{two} be the elements of S and P which are in one *other* set and path, respectively i. e. $S_{two} = S \cap (\cup_{S_i \neq S} S_i)$ and P_{two} is defined analogously. It may be noted that $S_{two} = supp(\mathcal{F}')$ and $P_{two} = supp(\mathcal{F}'^{\ell'})$. From Lemma 3.3.2 it follows that there is a bijection ϕ from S_{two} to P_{two} such that for each i, the image of $S_i \cap S$ under ϕ is $P_i \cap P$ (definition of feasibility). Moreover, this means $|S_{two}| = |P_{two}|$. By definition, $S_{priv} = S \setminus S_{two}$ and $P_{priv} = P \setminus P_{two}$. Thus $|S_{priv}| = |S \setminus S_{two}| = |P \setminus P_{two}| = |P_{priv}|$. □

Now we will prove the effectiveness of Algorithm 1. Lemma 3.3.7 proves that `filter common leaf` does not destroy the feasibility of the input, if it is indeed feasible. Following this, Lemma 3.3.8 and Lemma 3.3.9 show that if the input is an ICPPL the result of `filter common leaf` is also an ICPPL and they both have the same set of feasibility hypergraph isomorphisms.

Lemma 3.3.7. *In Algorithm 1, if input $(\mathcal{F}, \ell)$ is a feasible path labeling then at the end of jth iteration of the* **while** *loop, $j \geq 0$, $(\mathcal{F}_j, \ell_j)$ is a feasible path labeling.*

Proof. We will prove this by mathematical induction on the number of iterations. The base case $(\mathcal{F}_0, \ell_0)$ is feasible since it is the input itself which is given to be feasible. Assume the lemma is true till $j-1$th iteration i. e. every hypergraph isomorphism $\phi : supp(\mathcal{F}_{j-1}) \to V(T)$ that defines $(\mathcal{F}, \ell)$'s feasibility, is such that the induced path labeling on $\mathcal{F}_{j-1}$, say denoted by $p_{\phi[\mathcal{F}_{j-1}]}$, is equal to ℓ_{j-1}. We will prove that ϕ is also the bijection that makes $(\mathcal{F}_j, \ell_j)$ feasible. Note that $supp(\mathcal{F}_{j-1}) = supp(\mathcal{F}_j)$ since the new sets in $\mathcal{F}_j$ are created from basic set operations to the sets in $\mathcal{F}_{j-1}$ adding or removing no elements. For the same reason and ϕ being a bijection, it is clear that when applying the ϕ-induced path labeling on $\mathcal{F}_j$, $p_{\phi[\mathcal{F}_j]}(S_1 \setminus S_2) = p_{\phi[\mathcal{F}_{j-1}]}(S_1) \setminus p_{\phi[\mathcal{F}_{j-1}]}(S_2)$. Now observe that $\ell_j(S_1 \setminus S_2) = \ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2) = p_{\phi[\mathcal{F}_{j-1}]}(S_1) \setminus p_{\phi[\mathcal{F}_{j-1}]}(S_2)$. Thus the induced path labeling $p_{\phi[\mathcal{F}_j]} = \ell_j$. □

Lemma 3.3.8. *Given that the input to Algorithm 1 is an ICPPL, at the end of jth iteration, $j \geq 0$, of the* **while** *loop, the following invariants are maintained.*

I	$\ell_j(R)$ *is a path in* T,	*for all* $R \in \mathcal{F}_j$
II	$\lvert R\rvert = \lvert\ell_j(R)\rvert$,	*for all* $R \in \mathcal{F}_j$
III	$\lvert R \cap R'\rvert = \lvert\ell_j(R) \cap \ell_j(R')\rvert$,	*for all* $R, R' \in \mathcal{F}_j$
IV	$\lvert R \cap R' \cap R''\rvert = \lvert\ell_j(R) \cap \ell_j(R') \cap \ell_j(R'')\rvert$,	*for all* $R, R', R'' \in \mathcal{F}_j$

Proof. Proof is by induction on the number of iterations, j. In this proof, the term "new sets" will refer to the sets added to $\mathcal{F}_j$ in jth iteration in line 4 of Algorithm 1, $S_1 \cap S_2, S_1 \setminus S_2, S_2 \setminus S_1$ and its images in ℓ_j where $\ell_{j-1}(S_1)$ and $\ell_{j-1}(S_2)$ intersect and share a leaf. The invariants are true in the base case $(\mathcal{F}_0, \ell_0)$, since it is the input ICPPL. Assume the lemma is true till the $j-1$th iteration. Let us consider the possible cases for each of the above invariants for the jth iteration.

✠ *Invariant* I/II

I/IIa | *R is not a new set.* It is in $\mathcal{F}_{j-1}$. Thus trivially true by induction hypothesis.

I/IIb | *R is a new set.* If R is in $\mathcal{F}_j$ and not in $\mathcal{F}_{j-1}$, then it must be one of the new sets added in $\mathcal{F}_j$. In this case, it is clear that for each new set, the image under ℓ_j is a path since by definition the chosen sets S_1, S_2 are from $\mathcal{F}_{j-1}$ and due to the while loop condition, $\ell_{j-1}(S_1)$, $\ell_{j-1}(S_2)$ have a common leaf. Thus invariant I is proven.
Moreover, due to induction hypothesis of invariant III and the definition of l_j in terms of l_{j-1}, invariant II is indeed true in the jth iteration for any of the new sets. If $R = S_1 \cap S_2$, $|R| = |S_1 \cap S_2| = |\ell_{j-1}(S_1) \cap \ell_{j-1}(S_2)| = |\ell_j(S_1 \cap S_2)| = |\ell_j(R)|$. If $R = S_1 \setminus S_2$, $|R| = |S_1 \setminus S_2| = |S_1| - |S_1 \cap S_2| = |\ell_{j-1}(S_1)| - |\ell_{j-1}(S_1) \cap \ell_{j-1}(S_2)| = |\ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2)| = |\ell_j(S_1 \setminus S_2)| = |\ell_j(R)|$. Similarly if $R = S_2 \setminus S_1$.

✠ *Invariant* III

IIIa | *R and R′ are not new sets.* It is in $\mathcal{F}_{j-1}$. Thus trivially true by induction hypothesis.

IIIb | *Only one, say R, is a new set.* Due to invariant IV induction hypothesis, Lemma 3.3.3 and definition of ℓ_j, it follows that invariant III is true no matter which of the new sets R is equal to. If $R = S_1 \cap S_2$, $|R \cap R'| = |S_1 \cap S_2 \cap R'| = |\ell_{j-1}(S_1) \cap \ell_{j-1}(S_2) \cap \ell_{j-1}(R')| = |\ell_j(S_1 \cap S_2) \cap \ell_j(R')| = |\ell_j(R) \cap \ell_j(R')|$. If $R = S_1 \setminus S_2$, $|R \cap R'| = |(S_1 \setminus S_2) \cap R'| = |(\ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2)) \cap \ell_{j-1}(R')| = |\ell_j(S_1 \cap S_2) \cap \ell_j(R')| = |\ell_j(R) \cap \ell_j(R')|$. Similarly, if $R = S_2 \setminus S_1$. Note R' is not a new set.

IIIc | *R and R′ are new sets.* By definition, the new sets and their path images in path label ℓ_j are disjoint so $|R \cap R'| = |\ell_j(R) \cap \ell_j(R)| = 0$. Thus case proven.

✠ *Invariant* IV

This invariant is ensured at the end of every iteration due to Lemma 3.3.4 and Corollary 3.3.5.

□

Lemma 3.3.9. *If the input ICPPL $(\mathcal{F}, \ell)$ to Algorithm 1 is feasible, then the set of hypergraph isomorphism functions that defines $(\mathcal{F}, \ell)$'s feasibility is the same as the set that defines $(\mathcal{F}_j, \ell_j)$'s feasibility, if any.*

Proof. Since $(\mathcal{F}, \ell)$ is feasible, by Lemma 3.3.7 $(\mathcal{F}_j, \ell_j)$ for every iteration $j > 0$ is feasible. Also, every hypergraph isomorphism $\phi : supp(\mathcal{F}) \to V(T)$ that induces ℓ on $\mathcal{F}$ also induces ℓ_j on $\mathcal{F}_j$, i.e., $\ell_{\phi[\mathcal{F}_j]} = \ell_j$. Thus it can be seen that for all $x \in supp(\mathcal{F})$, for all

$v \in V(T)$, if $(x, v) \in \phi$ then $v \in \ell_j(S)$ for all $S \in \mathcal{F}_j$ such that $x \in S$. In other words, `filter common leaf` outputs a filtered path labeling that "preserves" hypergraph isomorphisms of the original path labeling.

□

As a result of `filter common leaf` each leaf v in T is such that there is exactly one set in $\mathcal{F}$ with v as a vertex in the path assigned to it. `filter fix leaf`, in Algorithm 2, identifies elements in $supp(\mathcal{F})$ whose images are leaves in a hypergraph isomorphism if one exists. Let $S \in \mathcal{F}$ be such that $\ell(S)$ is a path with leaf and $v \in V(T)$ is the unique leaf incident on it. We define a new path labeling ℓ_{new} such that $\ell_{new}(\{x\}) = \{v\}$ where x an arbitrary element from $S \setminus \bigcup_{\hat{S} \neq S} \hat{S}$. In other words, x is an element present in no other set in $\mathcal{F}$ except S. This is intuitive since v is present in no other path image under ℓ other than $\ell(S)$. The element x and leaf v are then removed from the set S and path $\ell(S)$ respectively. After doing this for all leaves in T, all path images in the new path labeling ℓ_{new} except leaf labels (a path that has only a leaf is called the *leaf label* for the corresponding single element hyperedge or set) are paths from a new pruned tree $T_0 = T \setminus \{v \mid v \text{ is a leaf in } T\}$. Algorithm 2 is now presented with details.

Algorithm 2 Leaf labeling from an ICPPL `filter fix leaf` $(\mathcal{F}, \ell, T)$

1: $\mathcal{F}_0 \leftarrow \mathcal{F}$, $\ell_0(S) \leftarrow \ell(S)$ for all $S \in \mathcal{F}_0$ | `Path images are such that no two path images share a leaf.`
2: $j \leftarrow 1$
3: **while** there is a leaf v in T and a unique $S_1 \in \mathcal{F}_{j-1}$ such that $v \in \ell_{j-1}(S_1)$ **do**
4: $\quad \mathcal{F}_j \leftarrow \mathcal{F}_{j-1} \setminus \{S_1\}$
5: $\quad$ for all $S \in \mathcal{F}_{j-1}$ such that $S \neq S_1$ set $\ell_j(S) \leftarrow \ell_{j-1}(S)$
6: $\quad X \leftarrow S_1 \setminus \bigcup_{S \in \mathcal{F}_{j-1}, S \neq S_1} S$
7: $\quad x \leftarrow$ arbitrary element from X
8: $\quad \mathcal{F}_j \leftarrow \mathcal{F}_j \cup \{\{x\}, S_1 \setminus \{x\}\}$
9: $\quad \ell_j(\{x\}) \leftarrow \{v\}$
10: $\quad \ell_j(S_1 \setminus \{x\}) \leftarrow \ell_{j-1}(S_1) \setminus \{v\}$
11: $\quad j \leftarrow j + 1$
12: **end while**
13: $\mathcal{F}' \leftarrow \mathcal{F}_j$, $\ell' \leftarrow \ell_j$
14: **return** $(\mathcal{F}', \ell')$

Suppose the input $(\mathcal{F}, \ell)$ is feasible, yet set X in Algorithm 2 is empty in some iteration of the **while** loop. Then clearly, the algorithm will have a problem in line 7. The following Lemma 3.3.10 shows that this cannot happen. Following that Lemma 3.3.11 shows that given an ICPPL input, the filtered output path labeling after `filter fix leaf` is also an ICPPL.

Lemma 3.3.10. *If the input ICPPL $(\mathcal{F}, \ell)$ to Algorithm 2 is feasible, then for all iterations $j > 0$ of the* **while** *loop, the set X will be non-empty.*

Proof. This directly follows from Lemma 3.3.6 since X is nothing but $S_{1_{priv}}$. □

Lemma 3.3.11. *In Algorithm 2, for all $j > 0$, at the end of the jth iteration of the* **while** *loop the four invariants given in Lemma 3.3.8 hold.*

Proof. Following Lemma 3.3.10, we know that set X will not be empty in any iteration of the **while** loop if input ICPPL $(\mathcal{F}, \ell)$ is feasible and thus ℓ_j is always computed for all $j > 0$. Also note that removing a leaf from any path keeps the new path connected. Thus invariant I is obviously true. In every iteration $j > 0$, we remove exactly one element x from one set S in $\mathcal{F}$ and exactly one vertex v which is a leaf from one path $\ell_{j-1}(S)$ in T. This is because x is exclusive to S and v is exclusive to $\ell_{j-1}(S)$. Due to this fact, it is clear that the intersection cardinality equations do not change, i.e., invariants II, III, IV remain true. □

We have seen two filtering algorithms above, namely, Algorithm 1 for `filter common leaf` and Algorithm 2 for `filter fix leaf` which when executed in that order result in a new ICPPL on the same universe U and target tree T. We also proved that if the input is indeed feasible, these algorithms executed in sequence give an ICPPL that preserves a non-empty subset of set of feasibility hypergraph isomorphisms of the original input. Now we present Algorithm 3 which uses these two filters to compute one such isomorphism.

Algorithm 3 computes a hypergraph isomorphism ϕ recursively using Algorithm 1 and Algorithm 2 and pruning the leaves of the input tree. In brief, it is done as follows. If the input is feasible, the output of `filter common leaf`, say $(\mathcal{F}_1, \ell_1)$, gives a new ICPPL which preserves all the original hypergraph isomorphisms. This ICPPL $(\mathcal{F}_1, \ell_1)$ is then given to `filter fix leaf` which gives the second filtered ICPPL, say $(\mathcal{F}_2, \ell_2)$ such that a feasible leaf labeling is in singleton elements of $\mathcal{F}_2$. These leaf labels are the elements in $supp(\mathcal{F})$ that map to leaves in T in at least one hypergraph isomorphism of the input ICPPL ℓ. It must be noted that at this point, the algorithm "chooses" one (or more - there could be more than one bijections that have the same leaf labels) hypergraph isomorphism ϕ', from the set of all isomorphisms resulting from the feasibility of the original input $(\mathcal{F}, \ell)$. However, if there exists one (i. e. input is feasible), it will be found – thus performing the test required in Feasible Tree Path Labeling.

Since their preimages have been found, all leaves in T are then pruned away. The leaf labels are removed from the path labeling ℓ_2 and the corresponding elements are removed from the corresponding sets in $\mathcal{F}_2$. It is now clear that we have a subproblem with a new hypergraph $\mathcal{F}'$, new tree path labeling ℓ' and target tree T'. The tree pruning algorithm is recursively called on $\mathcal{F}', \ell', T'$. The recursive call returns the bijection ϕ'' for the rest of the elements in $supp(\mathcal{F})$ which along with the leaf labels ϕ' computed earlier gives us a hypergraph isomorphism ϕ for the input $\mathcal{F}, \ell, T$. Lemma 3.3.12 proves the correctness

of this computation.

Algorithm 3 `get-hypergraph-isomorphism` $(\mathcal{F}, \ell, T)$

1: **if** T is empty **then**
2: **return** $\emptyset$
3: **end if**
4: $L \leftarrow \{v \mid v \text{ is a leaf in } T\}$
5: $(\mathcal{F}_1, \ell_1) \leftarrow$ `filter common leaf` $(\mathcal{F}, \ell, T)$
6: $(\mathcal{F}_2, \ell_2) \leftarrow$ `filter fix leaf` $(\mathcal{F}_1, \ell_1, T)$
7: $(\mathcal{F}', \ell') \leftarrow (\mathcal{F}_2, \ell_2)$
8: $\phi' \leftarrow \emptyset$
9: **for** every $v \in L$ **do**
10: $\phi'(x) \leftarrow v$ where $x \in \ell_2^{-1}(\{v\})$ | `Copy the leaf labels to a one to one function` $\phi' : supp(\mathcal{F}) \to L$
11: Remove $\{x\}$ and $\{v\}$ from $\mathcal{F}'$, ℓ' appropriately
12: **end for**
13: $T' \leftarrow T \setminus L$
14: $\phi'' \leftarrow$ `get-hypergraph-isomorphism` $(\mathcal{F}', \ell', T')$
15: $\phi \leftarrow \phi'' \cup \phi'$
16: **return** ϕ

Lemma 3.3.12. *If $(\mathcal{F}, \ell)$ is an ICPPL from a tree T and Algorithm 3, on input $(\mathcal{F}, \ell, T)$ returns a non-empty function ϕ, then ϕ is a hypergraph isomorphism $\phi : supp(\mathcal{F}) \to V(T)$ such that the ϕ-induced tree path labeling is equal to ℓ; $p_\phi = \ell$.*

Proof. It is clear that in the end of every recursive call to Algorithm 3, the function ϕ' is one-to-one involving all the leaves in the input target tree T (of the current recursive call). Moreover, by Lemma 3.3.9 and Lemma 3.3.10 it is consistent with the input tree path labeling ℓ (of the current recursive call). The tree pruning is done by only removing leaves in each call to the function and is done till the tree becomes empty. Thus the returned function $\phi : supp(\mathcal{F}) \to V(T)$ is a union of mutually exclusive one-to-one functions exhausting all vertices of the tree. In other words, it is a bijection from $supp(\mathcal{F})$ to $V(T)$ inducing the given path labeling ℓ and thus a hypergraph isomorphism. □

Finally we have Theorem 3.3.13 which puts together all that we saw so far in this section to prove that ICPPL is indeed a characterization of a feasible path labeling.

Theorem 3.3.13. *A path labeling $(\mathcal{F}, \ell)$ on tree T is feasible if and only if it is an ICPPL and Algorithm 3 with $(\mathcal{F}, \ell, T)$ as input returns a non-empty function.*

Proof. From Lemma 3.3.12, we know that if $(\mathcal{F}, \ell)$ is an ICPPL and if Algorithm 3 with $(\mathcal{F}, \ell, T)$ as input returns a non-empty function, then $(\mathcal{F}, \ell)$ is feasible. Now consider the case where $(\mathcal{F}, \ell)$ is feasible, i. e. there exists a hypergraph isomorphism ϕ such that $p_\phi = \ell$. Algorithm 3 then clearly returns a non-empty function because the target tree is non-empty. □

3.4 Solution to study group accommodation problem

In this section we run through the filtering and tree pruning algorithm seen in Section 3.3 on an example. The example problem is the study group accommodation problem posed in Section 1.2. The solution is depicted in Figure 3.2, Figure 3.3, Figure 3.4, Figure 3.5, Figure 3.6, Figure 3.7, Figure 3.8 and the figures along with their captions are self-explanatory.

In Figure 3.2, the original problem from Section 1.2 is restated more formally. The target tree is T with vertex set $\{1, 2, \ldots, 11\}$ on the left hand side along with the path system (hypergraph) – $\{7, 2, 6, 5\}, \{8, 2, 4\}, \{10, 6, 5, 3\}, \{9, 1, 5, 3, 11\}$. The hypergraph is $\mathcal{F}$ with hyperedges $\{\mathbf{Pa}, \mathbf{Pi}, \mathbf{Vi}, \mathbf{Ch}\}, \{\mathbf{Sn}, \mathbf{Wo}, \mathbf{Pi}\}, \{\mathbf{Ch}, \mathbf{Fr}, \mathbf{Vi}, \mathbf{Li}\}, \{\mathbf{Lu}, \mathbf{Sa}, \mathbf{Sc}, \mathbf{Ch}, \mathbf{Fr}\}$ shown in the right hand side. In subsequent illustrations, the gray boxes which contain the elements of universe U, will be assigned to nodes and removed from the right hand side figure and placed next to the vertex of T in the left hand side figure. The path labeling is implicitly given by the color-coded Venn diagrams (the order of listing of the hyperedges in given above is also done respective to the input path labeling).

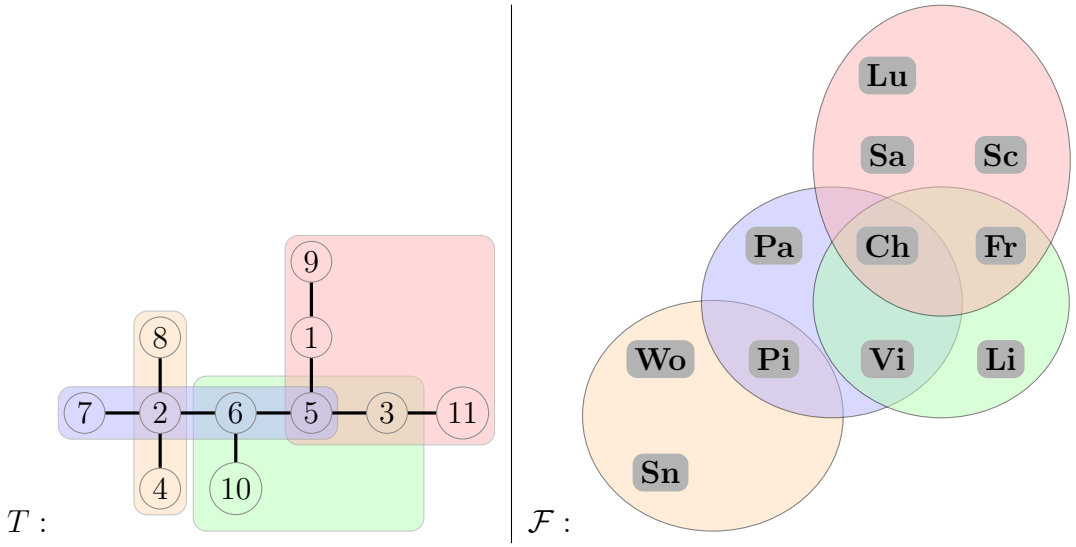


Figure 3.2: The problem is shown again here using the input described in Section 1.2.

Now see Figure 3.3 to see how the algorithm proceeds. Before that it can be observed in Figure 3.2, that none of the leaves are shared by more than one path. Hence, `filter common leaf` does not alter the hypergraphs. `filter fix leaf` finds the leaf assignments as shown – elements **Pa**, **Sn**, **Lu**, **Sc**, **Li**, **Wo** are assigned to vertices 7, 8, 9, 11, 10, 4 respectively. Computation of these leaf labels are deterministic except for **Lu**, **Sc** where the algorithm arbitrarily chooses them from $X = \{\mathbf{Lu}, \mathbf{Sa}, \mathbf{Sc}\}$ and for **Sn**, **Wo** arbitrarily choosen from $X = \{\mathbf{Sn}, \mathbf{Wo}\}$. Here we show only one such choice and clearly one could have chosen another leaf label (e. g. **Sa** on 9 or **Wo** on 8) to get a different feasible hypergraph isomorphism. After this, the leaves of T are pruned and the hypergraph updated accordingly by `get-hypergraph-isomorphism` subroutine – the pruned leaves

are shown with dotted edges and the corresponding Venn diagrams show the updated hypergraphs. ϕ_0 is this leaf assignment and the algorithm proceeds to solve the subproblem with the new hypergraphs and path labeling which is: $\{2,6,5\},\{2\},\{6,5,3\},\{1,5,3\}$ labeled with $\{\mathbf{Pi},\mathbf{Vi},\mathbf{Ch}\},\{\mathbf{Pi}\},\{\mathbf{Ch},\mathbf{Fr},\mathbf{Vi}\},\{\mathbf{Sa},\mathbf{Ch},\mathbf{Fr}\}$ respectively.

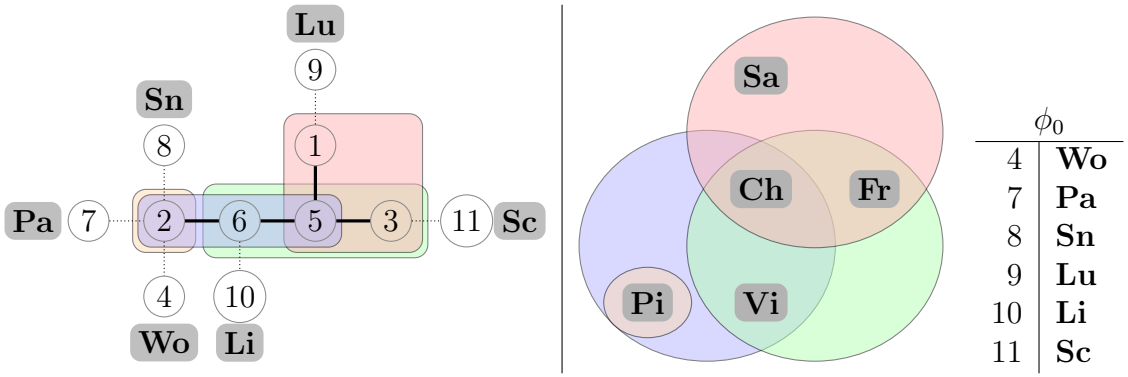


Figure 3.3: `filter fix leaf` finds leaf assignments as shown.

In the subproblem shown in Figure 3.3, it can be noted that there are two leaves which are shared by more than one paths – 2 and 3. Hence `filter common leaf` filters this out one by one. In Figure 3.4, we show the handling of leaf 2 which was shared by paths $\{2,6,5\}$ $(= P_1$, say) and $\{2\}(= P_2$, say). `filter common leaf` removes P_1 and adds $\{6,5\}$. Note that only one set is added and removed since $P_2 \subset P_1$. Accordingly, `filter common leaf` alters the hypergraph on the right hand side by removing and adding $\{\mathbf{Pi},\mathbf{Vi},\mathbf{Ch}\}$ and $\{\mathbf{Vi},\mathbf{Ch}\}$ respectively.

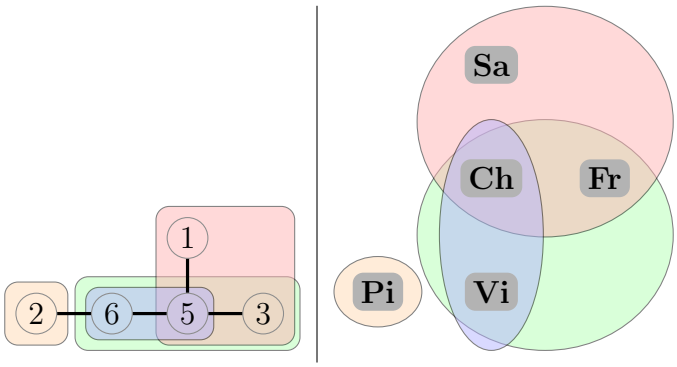


Figure 3.4: `filter common leaf` handling shared leaf 2.

After handling leaf 2, `filter common leaf` proceeds to handle leaf 3 which is shared by paths $\{1,5,3\}$ and $\{6,5,3\}$. This is illustrated in Figure 3.5. `filter common leaf` removes these two paths and adds their intersection and set differences – $\{5,3\}$, $\{1\}$ and $\{6\}$. Correspondingly, hyperedges $\{\mathbf{Sa},\mathbf{Ch},\mathbf{Fr}\}$ and $\{\mathbf{Vi},\mathbf{Ch},\mathbf{Fr}\}$ are removed and new hyperedges $\{\mathbf{Ch},\mathbf{Fr}\}$, $\{\mathbf{Sa}\}$ and $\{\mathbf{Vi}\}$ respectively are added. There are no more leaves shared and the algorithm proceeds to `filter fix leaf`.

Next, Figure 3.6 shows how `filter fix leaf` computes the leaf assignment ϕ_1 – $\mathbf{Pi},\mathbf{Sa},\mathbf{Fr}$ on 2,1,3 respectively. These assignments are deterministic since X consisted of only single elements for each of these leaves. After this the rest of the labeling is fairly obvious but we present it in Figure 3.7 for the sake of completeness.

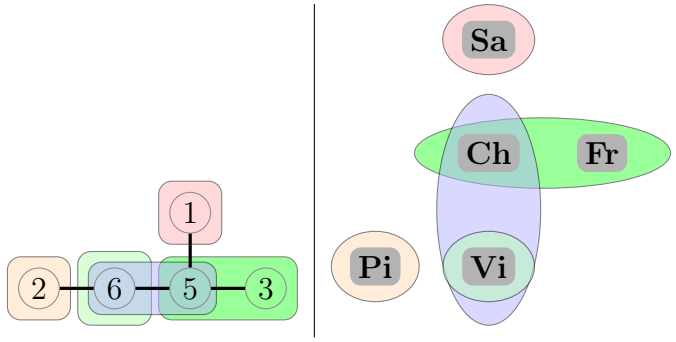


Figure 3.5: `filter common leaf` handles shared leaf 3.

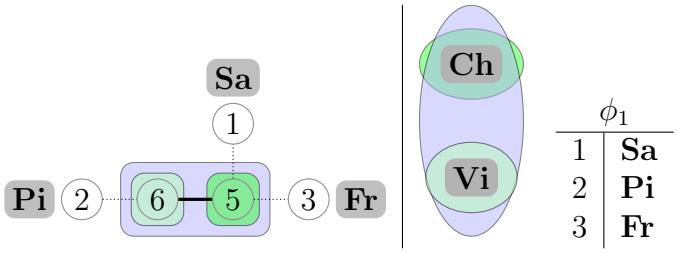


Figure 3.6: `filter fix leaf` computes the leaf assignment ϕ_1.

In Figure 3.7, the last part of the isomorphism is computed as the leaf label ϕ_2 which maps **Ch**, **Vi** to 5, 6 respectively. Figure 3.8 shows the final feasibility isomorphism which is the union of all the leaf labels. The feasibility hypergraph isomorphism ϕ is computed as the union of the leaf labels ϕ_0, ϕ_1, ϕ_2. One can easily verify that this isomorphism does indeed induce the path labeling given in the original input.

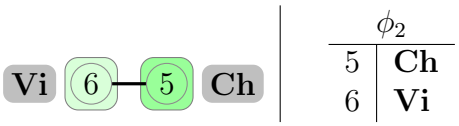
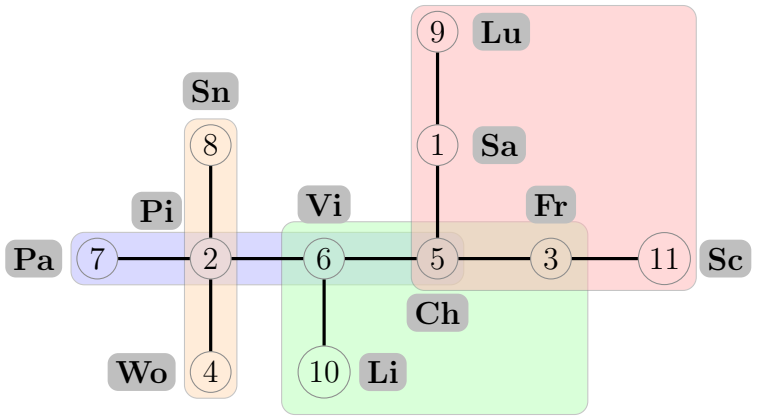


Figure 3.7: Last part of solution computed.

3.5 Computing feasible TPL with special target trees

Section 3.3 described properties that a TPL must have for it to be feasible. The next problem of interest is to test if a given hypergraph is a path hypergraph with respect to a given target tree [1]. In other words, the problem is to find out if a feasible tree path labeling exists from a given target tree for a given hypergraph. In this section we will see two special cases of this problem where the target tree is from a particular family of trees. The first one, where the tree is a path as described by the COMPUTE INTERVAL LABELING problem (see Section 3.1 for definition) in Section 3.5.1 is known to be equivalent to the well studied problem of consecutive-ones. The second one, where the tree is a k-subdivided tree as described by COMPUTE k-SUBDIVIDED STAR PATH LABELING (see

1. A larger problem would be to test if a given hypergraph is a path hypergraph where target tree is not given as input. This problem is not addressed in this thesis and is discussed as further research in Chapter 4.



$\phi = \phi_0 \cup \phi_1 \cup \phi_2$	
1	**Sa**
2	**Pi**
3	**Fr**
4	**Wo**
5	**Ch**
6	**Vi**
7	**Pa**
8	**Sn**
9	**Lu**
10	**Li**
11	**Sc**

Figure 3.8: The solution to the problem presented in Figure 3.2.

Section 3.1 for definition), has been solved using a polynomial time algorithm. The latter problem enforces some conditions on the hypergraph too which will be seen in Section 3.5.2.

3.5.1 Target tree is a Path

In Section 1.5 it is described how COP is a special case of TPL as described by the Compute Interval Labeling problem. This special case of TPL is where the target tree T is a path. All path labels can now be viewed as intervals assigned to the hyperedges in input hypergraph $\mathcal{F}$. It is shown, in [NS09], that the filtering algorithms outlined above need only preserve pairwise intersection cardinalities, and higher level intersection cardinalities are preserved by the Helly Property of intervals. Thus ICPIA as described in Section 2.2.5 is a sufficient and necessary condition for COP and this is solvable in polynomial time.

Moreover, this special case structure and its algorithm is used in the next section for finding tree path labeling from a k-subdivided star due to this tree's close relationship with intervals.

3.5.2 Target tree is a k-subdivided Star

In this section we consider the problem of assigning paths from a k-subdivided star (see Definition 3.2.6) T to a given set system $\mathcal{F}$ as described by the Compute k-subdivided Star Path Labeling problem. We consider $\mathcal{F}$ for which the overlap graph $\mathbb{O}(\mathcal{F})$ is connected. The overlap graph is well-known from the work of [KKLV10, NS09, Hsu02]. We use the notation in [KKLV10]. Recall from Section 3.2 that hyperedges S and S' are said to overlap, denoted by $S \between S'$, if S and S' have a non-empty intersection but neither of them is contained in the other. The overlap graph $\mathbb{O}(\mathcal{F})$ is a graph in which

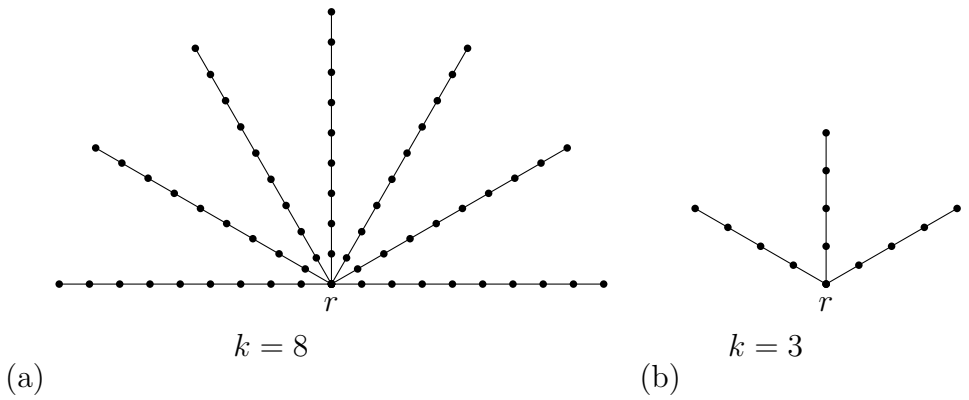


Figure 3.9: (a) 8-subdivided star with 7 rays (b) 3-subdivided star with 3 rays

the vertices correspond to the sets in $\mathcal{F}$, and the vertices corresponding to the hyperedges S and S' are adjacent if and only if they overlap. Note that the intersection graph of $\mathcal{F}$, $\mathbb{I}(\mathcal{F})$ is different from $\mathbb{O}(\mathcal{F})$ and $\mathbb{O}(\mathcal{F}) \subseteq \mathbb{I}(\mathcal{F})$. A connected component of $\mathbb{O}(\mathcal{F})$ is called an overlap component of $\mathcal{F}$. An interesting property of the overlap components is that any two distinct overlap components, say $\mathcal{O}_1$ and $\mathcal{O}_2$, are such that any two sets $S_1 \in \mathcal{O}_1$ and $S_2 \in \mathcal{O}_2$ are disjoint, or, w.l.o.g, all the sets in $\mathcal{O}_1$ are contained within one set in $\mathcal{O}_2$. This containment relation naturally determines a decomposition of the overlap components into rooted containment trees. We consider the case when there is only one rooted containment tree, and we first present our algorithm when $\mathbb{O}(\mathcal{F})$ is connected. It is easy to see that once the path labeling to the overlap component in the root of the containment tree is achieved, the path labeling to the other overlap components in the rooted containment tree is essentially finding a path labeling when the target tree is a path: each target path is a path that is allocated to sets in the root overlap component. Therefore, for the rest of this section, $\mathbb{O}(\mathcal{F})$ is a connected graph. We also assume that all hyperedges are of cardinality at most $k + 2$.

Recall from Section 3.2 that a k-subdivided star is a star with each edge subdivided k times. Therefore, a k-subdivided star has a central vertex which we call the *root*, and each root to leaf path is called a *ray*. First, we observe that by removing the root r from T, we get a collection of p vertex disjoint paths of length $k + 1$, p being the number of leaves in T. We denote the rays by $R_1, \ldots, R_p$, and the number of vertices in R_i, $i \in [p]$ is $k + 2$. Let $\langle v_{i1}, \ldots, v_{i(k+2)} = r \rangle$ denote the sequence of vertices in R_i, where v_{i1} is the leaf. Note that r is a common vertex to all R_i.

Description of the Algorithm. In this section the given hypergraph $\mathcal{F}$, the k-subdivided star and the root of the star are denoted by $\mathcal{O}$, T and vertex r, respectively. In particular, note that the vertices of $\mathcal{O}$ correspond to the sets in $\mathcal{F}$, and the edges correspond to the overlap relation.

For each hyperedge $X \in \mathcal{O}$, we will maintain a 2-tuple of non-negative numbers $\langle p_1(X), p_2(X) \rangle$. The numbers satisfy the property that $p_1(X) + p_2(X) \leq |X|$, and at the end of path labeling, for each X, $p_1(X) + p_2(X) = |X|$. This signifies the algorithm tracking the lengths

of subpaths of the path assigned to X from at most two rays. We also maintain another parameter called the *residue* of X denoted by $s(X) = |X| - p_1(X)$. This signifies the residue path length that must be assigned to X which must be from another ray. For instance, if X is labeled a path from only one ray, then $p_1(X) = |X|$, $p_2(X) = 0$ and $s(X) = 0$.

The algorithm proceeds in iterations, and in the i-th iteration, $i > 1$, a single hyperedge X that overlaps with a hyperedge that has been assigned a path is considered. At the beginning of each iteration hyperedges of $\mathcal{O}$ are classified into the following disjoint sets.

$\mathcal{L}_1^i$ *Labeled without r.* Those that have been labeled with a path which does not contain r in one of the previous iterations.
$\mathcal{L}_1^i = \{X \mid p_1(X) = |X| \text{ and } p_2(X) = 0 \text{ and } s(X) = 0, X \in \mathcal{O}\}$

$\mathcal{L}_2^i$ *Labeled with r.* Those that have been labeled with two subpaths of $\ell(X)$ containing r from two different rays in two previous iterations.
$\mathcal{L}_2^i = \{X \mid 0 < p_1(X), p_2(X) < |X| = p_1(X) + p_2(X) \text{ and } s(X) = 0, X \in \mathcal{O}\}$

$\mathcal{T}_1^i$ *Type 1 / partially labeled.* Those that have been labeled with one path containing r from a single ray in one of the previous iterations. Here, $p_1(X)$ denotes the length of the subpath of $\ell(X)$ that X has been so far labeled with.
$\mathcal{T}_1^i = \{X \mid 0 < p_1(X) < |X| \text{ and } p_2(X) = 0 \text{ and } s(X) = |X| - p_1(X), X \in \mathcal{O}\}$

$\mathcal{T}_2^i$ *Type 2 / not labeled.* Those that have not been labeled with a path in any previous iteration.
$\mathcal{T}_2^i = \{X \mid p_1(X) = p_2(X) = 0 \text{ and } s(X) = |X|, X \in \mathcal{O}\}$

The set $\mathcal{O}_i$ refers to the set of hyperedges $\mathcal{T}_1^i \cup \mathcal{T}_2^i$ in the ith iteration. Note that $\mathcal{O}_1 = \mathcal{O}$. In the ith iteration, hyperedges from $\mathcal{O}_i$ are assigned paths from T using the following rules. Also the end of the iteration, $\mathcal{L}_1^{i+1}, \mathcal{L}_2^{i+1}, \mathcal{T}_1^{i+1}, \mathcal{T}_2^{i+1}$ are set to $\mathcal{L}_1^i, \mathcal{L}_2^i, \mathcal{T}_1^i, \mathcal{T}_2^i$ respectively, along with some case-specific changes mentioned in the rules below.

I. **Iteration 1:** Let $S = \{X_1, \ldots, X_s\}$ denote the super-marginal hyperedges from $\mathcal{O}_1$. If $|S| = s \neq p$, then exit reporting failure. Else, assign to each $X_j \in S$, the path from R_j such that the path contains the leaf in R_j. This path is referred to as $\ell(X_j)$. Set $p_1(X_j) = |X|, p_2(X_j) = s(X_j) = 0$. Hyperedges in S are not added to $\mathcal{O}_2$ but are added to $\mathcal{L}_1^2$ and all other hyperedges are added to $\mathcal{O}_2$.

II. **Iteration i:** Let X be a hyperedge from $\mathcal{O}_i$ such that there exists $Y \in \mathcal{L}_1^i \cup \mathcal{L}_2^i$ and $X \between Y$. Further let $Z \in \mathcal{L}_1^i \cup L_2^i$ such that $Z \between Y$. If $X \in \mathcal{T}_2^i$, and if there are multiple Y candidates then any Y is selected. On the other hand, if $X \in \mathcal{T}_1^i$, then X has a partial path assignment, $\ell'(X)$ from a previous iteration, say from ray R_j. Then, Y is chosen such that $X \cap Y$ has a non-empty intersection with a ray

different from R_j. The key things that are done in assigning a path to X are as follows. The *end* of path $\ell(Y)$ where $\ell(X)$ would overlap is found, and then based on this the existence of a feasible assignment is decided. It is important to note that since $X \between Y$, $\ell(X) \between \ell(Y)$ in any feasible assignment. Therefore, the notion of the *end* at which $\ell(X)$ and $\ell(Y)$ overlap is unambiguous, since for any path, there are two end points.

(a) *End point of $\ell(Y)$ where $\ell(X)$ overlaps depends on $X \cap Z$:* If $X \cap Z \neq \emptyset$, then $\ell(X)$ has an overlap of $|X \cap Y|$ at that end of $\ell(Y)$ at which $\ell(Y)$ and $\ell(Z)$ overlap. If $X \cap Z = \emptyset$, then $\ell(X)$ has an overlap of $|X \cap Y|$ at that end of $\ell(Y)$ where $\ell(Y)$ and $\ell(Z)$ do not intersect.

(b) *Any path of length $s(X)$ at the appropriate end contains r:* If $X \in \mathcal{T}_1^i$ then after finding the appropriate end as in step IIa this the unique path of length $s(X)$ should end at r. If not, we exit reporting failure. Else, $\ell(X)$ is computed as union of $\ell'(X)$ and this path. If any three-way intersection cardinality is violated with this new assignment, then exit, reporting failure. Otherwise, X is added to $\mathcal{L}_2^{i+1}$. On the other hand, if $X \in \mathcal{T}_2^i$, then after step IIa, $\ell(X)$ or $\ell'(X)$ is unique up to the root and including it. Clearly, the vertices $\ell(X)$ or $\ell'(X)$ contains depends on $|X|$ and $|X \cap Y|$. If any three way intersection cardinality is violated due to this assignment, exit, reporting failure. Otherwise, $p_1(X)$ is updated as the length of the assigned path, and $s(X) = |X| - p_1(X)$. If $s(X) > 0$, then X is added to $\mathcal{T}_1^{i+1}$. If $s(X) = 0$, then X is added to $\mathcal{L}_1^{i+1}$.

(c) *The unique path of length $s(X)$ overlapping at the appropriate end of Y does not contain r:* In this case, $\ell(X)$ is updated to include this path. If any three way intersection cardinality is violated, exit, reporting failure. Otherwise, update $p_1(X)$ and $p_2(X)$ are appropriate, X is added to $\mathcal{L}_1^{i+1}$ or $\mathcal{L}_2^{i+1}$, as appropriate.

Proof of Correctness and Analysis of Running Time: It is clear that the algorithm runs in polynomial time, as at each step, at most three-way intersection cardinalities need to be checked. Further, finding super-marginal hyperedges can also be done in polynomial time, as it involves considering the overlap regions and checking if the inclusion partial order contains a single minimal element. In particular, once the super-marginal edges are identified, each iteration involves finding the next hyperedge to consider, and testing for a path to that hyperedge. To identify the next hyperedge to consider, we consider the breadth first layering of the hyperedges with the zeroth layer consisting of the super-marginal hyperedges. Since $\mathcal{O}$ is connected, it follows that all hyperedges of $\mathcal{O}$ will be considered by the algorithm. Once a hyperedge is considered, the path to be assigned to it can also be computed in constant time. In particular, in the algorithm the path to be assigned to X depends on $\ell(Y), \ell(Z)$, $s(X)$ and the presence or absence of r in the candidate partial path $\ell'(X)$. Therefore, once the super-marginal edges are identified,

the running time of the algorithm is linear in the size of the input. By the technique used for constructing prime matrices [Hsu02], the super-marginal edges can be found in linear time in the input size. Therefore, the algorithm can be implemented to run in linear time in the input size.

The proof of correctness uses the following main properties:

1. The k-subdivided star has a very symmetric structure. This symmetry is quantified based on the following observation – either there are no feasible path labelings of $\mathcal{O}$ using paths from T, or there are exactly $p!$ feasible path labelings. In other words, there is either no feasible assignment, or effectively a unique assignment modulo symmetry.

2. The p super-marginal hyperedges, if they exist, will each be assigned a path from distinct rays, and each such path contains the leaf.

3. For a candidate hyperedge X, the partial path assignment $\ell'(X)$ is decided by its overlap with $\ell(Y)$ and cardinality of intersection with $\ell(Z)$.

These properties are formalized as follows:

Lemma 3.5.1. *If $X \in \mathcal{F}$ is super-marginal and ℓ is a feasible tree path labeling to tree T, then $\ell(X)$ will contain a leaf in T.*

Proof. Suppose $X \in \mathcal{F}$ is super-marginal and $(\mathcal{F}, \ell)$ is a feasible path labeling from T. Assume $\ell(X)$ does not have a leaf. Let R_i be one of the rays (or the only ray) $\ell(X)$ is part of. Since X is in a connected overlap component, there exists $Y_1 \in \mathcal{F}$ and $X \nsubseteq Y_1$ such that $Y_1 \between X$ and Y_1 has at least one vertex closer to the leaf in R_i than any vertex in X. Similarly with the same argument there exists $Y_2 \in \mathcal{F}$ with same subset and overlap relation with X except it has has at least one vertex farther away from the leaf in R_i than any vertex in X. Clearly $Y_1 \cap X$ and $Y_2 \cap X$ cannot be part of same inclusion chain which contradicts that assumption X is super-marginal. Thus the claim is proved. □

Lemma 3.5.2. *If $\mathcal{O}$ does not have any super-marginal edges, then in any feasible path labeling ℓ of $\mathcal{O}$ with paths from T is such that, for any hyperedge X for which $\ell(X)$ contains a leaf, $|X| \geq k + 3$.*

Proof. The proof of this lemma is by contradiction. Let X be a hyperedges such that $|X| \leq k+2$ and that $\ell(X)$ has a leaf. This implies that the overlap regions with X, which are captured by the overlap regions with $\ell(X)$, will form a single inclusion chain. This shows that X is a marginal hyperedge which contradicts the assumption that $\mathcal{O}$ does not have super-marginal hyperedges. □

This lemma is used to prove the next lemma for the case when for all $X \in \mathcal{O}$, $|X| \leq k+2$. The proof is left out as it just uses the previous lemma and the fact that the hyperedges in X have at most $k+2$ elements.

Lemma 3.5.3. *If there is a feasible path labeling for $\mathcal{O}$ in T, then there are exactly p super-marginal hyperedges.*

These lemmas now are used to prove the following theorem.

Theorem 3.5.4. *Given $\mathcal{O}$ and a k-subdivided star T, the above algorithm decides correctly if there is a feasible path labeling ℓ.*

Proof. Outline. If the algorithm outputs a path labeling ℓ, then it is clear that it is an ICPPL. The reason is that the algorithm checks that three-way intersection cardinalities are preserved in each iteration which ensures ICPPL Property iii. Moreover, it is clear that $\ell(X)$ for any $X \in \mathcal{O}$ is computed by maintaining ICPPL Property i and ICPPL Property ii. For such a labeling ℓ, the proof that it is feasible is by induction on k. What needs to be shown is that Algorithm 3 successfully runs on input $(\mathcal{O}, \ell)$. In base case $k = 0$, T is a star. Also every set is at most size 2 ($k+2$) size and thus overlaps are at most 1. If two paths share a leaf in `filter common leaf` one must be of length 2 and the other of length 1. Thus the exit condition is not met. Further, it is also clear that the exit condition in `filter fix leaf` is also not met. Thus claim proven for base case. Now assume the claim to be true when target tree is a $(k-1)$-subdivided star. Consider the case of a k-subdivided star. We can show that after `filter common leaf` and one iteration of a modified `filter fix leaf` *all* leaves are assigned pre-images. Removing the leaves from T and the pre-images from support of $\mathcal{O}$, results in an ICPPL to a $(k-1)$-subdivided star. Now we apply the induction hypothesis, and we get an isomorphism between the hypergraphs $\mathcal{O}$ and $\mathcal{O}^{\ell}$.

In the reverse direction if there is a feasible path labeling ℓ, then we know that ℓ is unique up to isomorphism. Therefore, again by induction on k it follows that the algorithm finds ℓ. □

3.6 TPL on arbitrary trees

In this section we describe an algorithm for the Compute Feasible Tree Path Labeling problem where the target tree can be any arbitrary tree. In essence, we provide a characterization for path hypergraphs (see Section 3.2.2 for definition). We first study some properties of the overlap components of the input hypergraph and introduce some theory in that area. This is in the same vein as the theory used in [Hsu02, NS09] but we extend it to TPL. We use this theory to decompose Compute Feasible Tree Path

Labeling into subproblems. Each subproblem is on a sub-hypergraph of the input, in which for each hyperedge there is another hyperedge in the sub-hypergraph that overlaps with it. These sub-hypergraphs are called overlap components (see Section 3.2.3 for definition). As a consequence of this characterization, aside from the overlap component subproblems, we identify two other subproblems that must be solved to obtain an ICPPL. The inefficiency of our algorithm in terms of polynomial time solvability comes from these two subproblems. We leave these subproblems open to be solved in P and only mention the obvious brute force method of solution.

The two subproblems Find Overlap Component Partition Subtrees and Find MUB Feasible TPL are informally defined as follows. The precise definition is given after setting up the theory and notations required in the following paragraphs.

1. Find Overlap Component Partition Subtrees- *Dividing the tree T to subtrees for overlap component partitions:* Using the overlap partial ordering of hyperedges in $\mathcal{F}$, the hyperedges can be divided into partitions of sub-hypergraphs (called partitions of prime submatrices) that are disjoint from each other. Being disjoint they must be assigned to disjoint subtrees of T such that Find MUB Feasible TPL can be solved successfully and the other sub-hypergraphs in the partition will have an ICPIA in accordance with the overlap property. Finding the right partition of T into subtrees is the first subproblem.

2. Find MUB Feasible TPL- *Finding ICPPL for mub of each sub-hypergraph partition:* In each partition of sub-hypergraph, due to the overlap partial ordering, there will be an *mub* sub-hypergraph that contains all the other sub-hypergraphs in that partition. If T_i is the subgraph assigned to this partition by solving Find Overlap Component Partition Subtrees, T_i is essentially the tree for the *mub*'s TPL. Computing an ICPPL for this *mub* sub-hypergraph from T_i is the second subproblem.

As described by Definition 2.2.3 a set system or hypergraph can be concisely represented by a binary matrix such that the row indices of the matrix denote the vertex set of the hypergraph and each column bijectively correspond to a hyperedge. And thus the ideas of feasible TPL and ICPPL are also defined for a binary matrix. If a TPL $(\mathcal{F}, \ell, T)$ is feasible, i. e. it is an ICPPL (see Section 3.3) and if M is the representative matrix for hypergraph $\mathcal{F}$, then we say that M has an ICPPL and vice versa.

Consider the overlap graph and overlap components of $\mathbb{O}(\mathcal{F})$ defined in Section 3.2.3. We use this to decompose M as described in [Hsu02, NS09]. A *prime submatrix* of M is defined as the matrix formed by a set of columns of M which correspond to an overlap component. Let us denote the prime submatrices of M by $M_1, \ldots, M_p$, each corresponding to one of the p overlap components of $\mathcal{F}$. Clearly, two distinct prime

submatrices have mutually exclusive sets of columns of M. Let $col(M_i)$ be the set of columns in the submatrix M_i. The support of a prime submatrix M_i and support of a set of prime submatrices X are defined as follows. Note that for each i, $supp(M_i) \subseteq U$ where U is the vertex set of hypergraph $\mathcal{F}$.

$$supp(M_i) = \bigcup_{j \in col(M_i)} S_j$$
$$supp(X) = \bigcup_{M \in X} supp(M)$$

Consider the binary relation $\preccurlyeq$ defined on the set of prime submatrices $\mathbb{P} = \{M_i \mid i \in [p]\}$ as follows.

$$\begin{aligned} \preccurlyeq = &\{(M_i, M_j) \mid \exists S \in M_i \text{ and } \exists S' \in M_j \text{ such that } S \subseteq S'\} \\ &\cup \{(M_i, M_i) \mid i \in [p]\} \end{aligned}$$

This relation is the same as that defined in [NS09]. The prime submatrices and the above relation can be defined for any hypergraph since an overlap graph is defined for any hypergraph. We will use this structure of prime submatrices to present our results on an ICPPL for a hypergraph $\mathcal{F}$. Now we present a few lemma and a theorem that show that $\preccurlyeq$ is a partial order. These results are from [NS09] but are independent of the COP property of M and is valid for any binary matrix M except Theorem 3.6.5 which we subsequently extend in Theorem 3.6.6 to accommodate ICPPL. Note that an alternate notation for $(M_i, M_j) \in \preccurlyeq$ is $M_i \preccurlyeq M_j$.

Lemma 3.6.1 ([NS09, Lem. 3]). *If $M_i \preccurlyeq M_j$, then there is a set $S' \in M_j$ such that for each $S \in M_i$, $S \subseteq S'$.*

Lemma 3.6.2 ([NS09, Lem. 4]). *For each pair of prime submatrices M_i, M_j, either $M_i \not\preccurlyeq M_j$ or $M_j \not\preccurlyeq M_i$.*

Lemma 3.6.3 ([NS09, Lem. 5]). *If $M_i \preccurlyeq M_j$ and $M_i \preccurlyeq M_k$, then $M_i \preccurlyeq M_k$.*

Lemma 3.6.4 ([NS09, Lem. 6]). *If $M_i \preccurlyeq M_j$ and $M_i \preccurlyeq M_k$, then either $M_j \preccurlyeq M_k$ or $M_k \preccurlyeq M_j$.*

Theorem 3.6.5 ([NS09, Th. 4]). *Let M be a binary matrix and $\mathbb{P}$ be the set of prime submatrices of M. Then the following hold true.*

i. *$\preccurlyeq$ is a partial order on $\mathbb{P}$.*

ii. *$\preccurlyeq$ uniquely partitions $\mathbb{P}$ into $\mathbb{P}_1, \mathbb{P}_2 \ldots, \mathbb{P}_r$ for some $r \geq 1$ as follows.*

iii. *If M has COP, then for each $i \in [r]$, $\preccurlyeq$ induces a total order in $\mathbb{P}_i$.*

[NS09] does not explicitly mention how the partitioning of $\mathbb{P}$ is done in Theorem 3.6.5 and merely mention that the total order in (iii) is obvious. We call this partitioning in Theorem 3.6.5 point (ii) as *chain partitioning*. We explain what chain partitioning is in the following text. For that, first we construct the Hasse diagram of $\preccurlyeq$ which we observe is a collection of in-trees in the following Theorem 3.6.6. Each of these in-trees have disjoint vertex sets, which in this case would be disjoint sets of prime submatrices $\mathbb{P}_i$, $i \in [r]$ for some $r > 0$. For each $\mathbb{P}_i$, the in-tree is rooted at a maximal upper bound under $\preccurlyeq$. The in-trees are specified by selecting the appropriate edges from the Hasse diagram associated with $\preccurlyeq$. This is defined in Theorem 3.6.6 before which we see the covering relation for $\preccurlyeq$ defined below.

$$\begin{aligned} \preccurlyeq_c &= \{(M_i, M_j) \mid M_i \preccurlyeq M_j \text{ such that } \nexists M_k \text{ s.t. } M_i \preccurlyeq M_k,\ M_k \preccurlyeq M_j\} \\ &\cup \{(M_i, M_i), i \in [p]\} \end{aligned}$$

It is easy to see that $\preccurlyeq_c$ is the covering relation of the partial order $\preccurlyeq$.

Theorem 3.6.6. *Let M be a binary matrix and $\mathbb{P}$ be the set of prime matrices of M. The partial order $\preccurlyeq$ and its covering relation $\preccurlyeq_c$ are as defined earlier. Then the directed graph $X = (\mathbb{P}, \preccurlyeq_c)$ whose vertex set is the set of prime submatrices and the edges are given by the covering relation $\preccurlyeq_c$, is a vertex disjoint collection of in-trees and the root of each in-tree is a maximal upper bound in $\preccurlyeq$.*

Proof. To observe that X is a collection of in-trees, we observe that for vertices corresponding to maximal upper bounds, no outgoing edge is present in $\preccurlyeq_c$. Secondly, for each other element, exactly one outgoing edge is chosen, and for the minimal lower bound, there is no incoming edge. Consequently, X is acyclic, and since each vertex has at most one edge leaving it, it follows that X is a collection of vertex disjoint in-trees, and for each in-tree, the root is a maximal upper bound in $\preccurlyeq$. □

We call the partitioning of set of prime matrices $\mathbb{P}$ given by Theorem 3.6.6 as *containment partitioning*. The term signifies the containment property described in Lemma 3.6.1 and the fact that each containment parts are disjoint from one another.

Let the containment partition of set of prime matrices $\mathbb{P}$ be:

$$\{\mathbb{P}_i \mid i \in [r], \mathbb{P}_i \text{ is the vertex set of an in-tree of } X\}$$

Further, the vertices of each in-tree can be layered based on their distance from the root, the root being the maximal upper bound. The root is considered to be at level zero. For $j \geq 0$, let $\mathbb{P}_{i,j}$ denote the set of prime matrices in level j of in-tree $\mathbb{P}_i$. The set $\mathbb{P}_{i,0}$ has only one prime submatrix which is the maximal upper bound in $\mathbb{P}_i$. Hence $\mathbb{P}_{i,0}$ is

denoted by $mub(\mathbb{P}_i)$.

Now we describe how the chain partitioning alluded to in Theorem 3.6.5 is obtained. Each in-tree described above is decomposed by the following rules into chains. Traverse the tree level by level from bottom to top. At each level, if there is a node with degree greater than 2, the chain that contains this node and all its descendants is one chain part. Remove this chain from the tree and recurse till reaching the root. Now it is clear why each of these partitions are totally ordered – it is so because they are chains of the Hasse diagram. Now we continue with the description of $\preccurlyeq$ for TPLs.

The next Lemma 3.6.7 and Lemma 3.6.8 show why it is necessary to assign a subtree of the target tree only to the *mub* of each in-tree and all other matrices reduce to interval assignment problems.

Lemma 3.6.7. *Let M be a matrix and let X be the directed graph whose vertices are in correspondence with the prime submatrices of M. Further let $\{\mathbb{P}_i \mid i \in [r]\}$ be the partition of X into in-trees as defined above. Then, matrix M has an ICPPL on target tree T if and only if T can be partitioned into vertex disjoint subtrees $\{T_i \mid i \in [r]\}$ such that, for each $i \in [r]$, the set of prime submatrices $\mathbb{P}_i$ has an ICPPL on subtree T_i.*

Proof. Let us consider the reverse direction first. Let us assume that T can be partitioned into $T_1, \ldots, T_r$ such that for each $i \in [r]$, the set of prime submatrices $\mathbb{P}_i$ has an ICPPL on T_i. It is clear from the properties of the partial order $\preccurlyeq$ that this naturally yield an ICPPL of M on T. The main property used in this inference is that for each $1 \leq i \neq j \leq r$, $supp(X_i) \cap supp(X_j) = \emptyset$. To prove the forward direction, we show that if M has an ICPPL, say ℓ, on T then there exists a partition of T into vertex disjoint subtree $T_1, \ldots, T_r$ such that for each $i \in [r]$, the set of prime submatrices $\mathbb{P}_i$ has an ICPPL on T_i. For each $i \in [r]$, we define based on ℓ a subtree T_i corresponding to X_i. We then argue that the subtrees thus defined are vertex disjoint and prove the claim. In partition $\mathbb{P}_i$, consider prime submatrix $mub(\mathbb{P}_i)$ and the paths labeled under ℓ to the hyperedges in the prime submatrix $mub(\mathbb{P}_i)$. Since the component in $\mathbb{O}(M)$ corresponding to this matrix (this notation is analogous to overlap graph $\mathbb{O}(\mathcal{F})$ for a hypergraph $\mathcal{F}$) is a connected component, it follows that the paths labeled to hyperedges in this prime submatrix form a (connected) subtree of T. We call this subtree T_i. All other prime submatrices in $\mathbb{P}_i$ are assigned merely paths in T_i. This follows from Lemma 3.6.1 along with the facts that ℓ is an ICPPL and $M_x \preccurlyeq mub(\mathbb{P}_i)$ for any prime submatrix $M_x \in \mathbb{P}_i$. Secondly, for each $1 \leq i \neq j \leq r$, $supp\,(\mathbb{P}_i) \cap supp\,(\mathbb{P}_j) = \emptyset$ and ℓ is an ICPPL. Thus it follows that T_i and T_j are vertex disjoint. Finally, since $|U| = |V(T)|$, it follows that $T_1, \ldots, T_r$ is indeed a partition of T. Thus the claim is proven. □

The essence of the following lemma is that an ICPPL only needs to be assigned to the prime submatrix corresponding to the root of each in-tree, and all the other prime sub-

matrices only need to have an Intersection Cardinality Preserving Interval Assignments (ICPIA). Recall, an ICPIA is an assignment of intervals to sets such that the cardinality of an assigned interval is same as the cardinality of the interval, and the cardinality of intersection of any two sets is same as the cardinality of the intersection of the corresponding intervals. It is shown in [NS09] that the existence of an ICPIA is a necessary and sufficient condition for a matrix to have COP. We present the pseudocode to test if M has an ICPPL in T.

Lemma 3.6.8. *Let M be a matrix, X be the directed graph whose vertices are in correspondence with the prime submatrices of M and $\{\mathbb{P}_1, \ldots, \mathbb{P}_r\}$ be the partition of X into in-trees by $\preccurlyeq$ defined earlier. Let T be the target tree and let $\{T_1, \ldots, T_r\}$ be a given partition of T into vertex disjoint subtrees. Then, for each $1 \leq i \leq r$, the set of prime submatrices $\mathbb{P}_i$ has an ICPPL in T_i if and only if the prime submatrix $mub(\mathbb{P}_i)$ has an ICPPL on T_i and all other matrices in $\mathbb{P}_i$ have an ICPIA.*

Proof. The proof is based on the fact that $\preccurlyeq$ is a partial order and X is a directed graph which is a disjoint union of in-trees. Each edge in the in-tree is a containment relationship among the supports of the corresponding submatrices. Therefore, any ICPPL to a prime submatrix that is not the *mub* is contained in a path assigned to the sets in the parent matrix. This follows from Lemma 3.6.1 and since $M_x \preccurlyeq mub(\mathbb{P}_i)$ for any prime submatrix $M_x \in \mathbb{P}_i$. Consequently, any ICPPL to the prime submatrix that is not $mub(\mathbb{P}_i)$ is an ICPIA, and any ICPIA can be used to construct an ICPPL to the matrices corresponding to nodes in $\mathbb{P}_i$ provided $mub(\mathbb{P}_i)$ has an ICPPL in T_i. □

Lemma 3.6.7 and Lemma 3.6.8 point out two algorithmic challenges in finding an ICPPL for a given hypergraph $\mathcal{F}$ on a target tree T, namely subproblems Find Overlap Component Partition Subtrees and Find MUB Feasible TPL respectively.

1. Find Overlap Component Partition Subtrees- Finding X and its partition into in-trees $\{\mathbb{P}_1, \ldots, \mathbb{P}_r\}$, can be done in polynomial time. On the other hand, as per Lemma 3.6.7 we need to partition T into vertex disjoint subtrees $\{T_1, \ldots, T_r\}$ such that for each $i \in [r]$, the prime submatrices in $\mathbb{P}_i$ have an ICPPL in T_i (to be accurate, the prime submatrix $mub(\mathbb{P}_i)$ has an ICPPL in T_i and the rest of the prime submatrices have an ICPIA as claimed in Lemma 3.6.8). This seems to be a challenging step, and it must be remarked that this step is easy when T itself is a path, as each individual T_i would be sub-paths.

2. Find MUB Feasible TPL- The second algorithmic challenge is identified by Lemma 3.6.8 which is to compute an ICPPL from the chosen subtree T_i to the matrix associated with $mub(\mathbb{P}_i)$.

The formal definitions of the problems are given below.

FIND OVERLAP COMPONENT PARTITION SUBTREES

Input	A hypergraph $\mathcal{F}$ with its in-tree partitions $\{\mathbb{P}_1, \ldots, \mathbb{P}_r\}$ and a tree T.
Question	Compute a partition of T into subtrees $\{T_1, \ldots, T_r\}$ such that there exists an ICPPL for $mub(\mathbb{P}_i)$ from subtree T_i for all $i \in [r]$ and there exists an ICPIA for all other prime submatrices in $\mathbb{P}_i$ as claimed in Lemma 3.6.8.

FIND MUB FEASIBLE TPL

Input	An in-tree partition $\mathbb{P}_i$ of a hypergraph $\mathcal{F}$ and a subtree T_i of a tree T.
Question	Compute a feasible TPL for $mub(\mathbb{P}_i)$ from subtree T_i.

Figure 3.10 and Figure 3.11 illustrate the containment partitions of a binary matrix and the claim of Lemma 3.6.8. In Figure 3.10, the feasibility bijection is explicitly shown by labeling the rows (which represents elements in hypergraph's universe) with the vertices of the target tree T, $v_1, \ldots, v_{12}$. M decomposed into prime submatrices $M_1, \ldots, M_6$ which is collectively represented by $\mathbb{P}$. It can be verified that they correspond to the overlap components in $\mathbb{O}(\mathcal{F})$. The containment partitions of $\mathbb{P}$ are $\mathbb{P}_1, \mathbb{P}_2$ as shown. Note that the empty cells of the matrix represent 0s – they are left empty to make the containment partitions more obvious. Figure 3.11 shows the target tree of this TPL. The rectangles depict the tree paths allocated to the columns in the *mub* of M's prime matrices, namely M_1 and M_4. The rest of the columns are allocated intervals illustrating Lemma 3.6.8.

Note that in this example the matrix has been explicitly labeled with the vertices of the tree which indicates the solution but arriving at this solution poses the above mentioned challenges. Challenge 1 in this example can be seen as follows. Figure 3.11 shows that $\mathbb{P}_1$ has been allocated the top subtree and $\mathbb{P}_2$ has been allocated the bottom subtree. Computationally this is not an easy decision. There is no obvious indication (as far as we know) that the tree must be decomposed at edge (v_3, v_{11}). If this was given, then one can make the decision based on the difference between the subtrees, namely sizes of the containment parts and the subtrees and conclude that $\mathbb{P}_1$ must be allocated the subtree $\{v_1, \ldots, v_7\}$ and $\mathbb{P}_2$ to the rest. What if the subtrees were of the same sizes, say vertex v_{14} in T, the fourteenth row in M and column S_9 in M were not present? One has to try to solve challenge 2 given above to conclude that the bottom (top) subtree indeed does not give an ICPPL to $\mathbb{P}_1$ ($\mathbb{P}_2$) and thus $\mathbb{P}_1$ ($\mathbb{P}_2$) but be allocated the top (bottom) subtree. Solving challenge 2 is not easy (as far as we know) either. By brute force, one can try all possible path labeling and check for ICPPL. If all of them fail, one must backtrack to challenge 1 and try another subtree for the containment part $\mathbb{P}_i$.

Algorithm 4 consolidates all that we described in this section and gives the procedure to solve COMPUTE FEASIBLE TREE PATH LABELING. It may be noted that all operations other than in line 5 and line 7 are known to be polynomial time solvable.

	M											
$X:$	$\mathbb{P}_1$						$\mathbb{P}_2$					
$\mathbb{P}:$	M_1			M_2		M_3	M_4				M_5	M_6
$\mathcal{F}:$	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}	S_{11}	S_{12}
v_1	0	0	**1**	0	0	0						
v_2	0	**1**	0	0	0	0						
v_3	**1**	**1**	**1**	0	0	0						
v_4	**1**	0	**1**	0	0	0						
v_5	**1**	0	0	**1**	**1**	0						
v_6	**1**	0	0	**1**	**1**	**1**						
v_7	**1**	0	0	**1**	0	**1**						
v_8							**1**	**1**	0	0	0	0
v_9							**1**	**1**	**1**	0	0	0
v_{10}							**1**	0	0	0	0	0
v_{11}							**1**	0	0	**1**	**1**	**1**
v_{12}							**1**	0	0	**1**	**1**	**1**
v_{13}							**1**	0	0	**1**	**1**	0
v_{14}							0	0	**1**	0	0	0
v_{15}							0	0	0	**1**	0	0

Figure 3.10: Partial order on prime submatrices. The table illustrates the structural properties of prime matrices and partial order $\preccurlyeq$. A hypergraph $\mathcal{F}$ with FTPL has hyperedges $S_1, \ldots, S_{12}$ which are represented by the binary matrix M. Empty entries denote zeros.

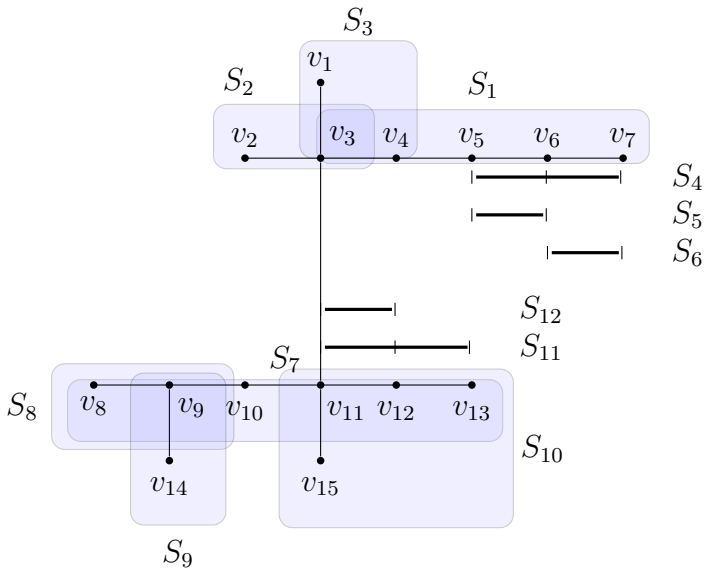


Figure 3.11: This graph is the target tree for the matrix in Fig. 3.10 with each column $S_i, 1 \leq i \leq 12$ shown as tree paths and intervals.

Algorithm 4 Algorithm to find an ICPPL for a matrix M on tree T: $main\_ICPPL(M, T)$

1: Identify the prime submatrices. This is done by constructing the strict overlap graph and identify connected components. Each connected component yields a prime submatrix.
2: Construct the partial order $\preccurlyeq$ on the set of prime submatrices.
3: Construct the partition $\mathbb{P}_1, \ldots, \mathbb{P}_r$ of the prime submatrices induced by $\preccurlyeq$
4: For each $1 \leq i \leq r$, check if all matrices except $mub(\mathbb{P}_i)$ has an ICPIA. If a matrix does not have ICPIA exit with a negative answer. To check for the existence of ICPIA, use the result in [NS09].
5: Find a partition of $T_1, \ldots, T_r$ such that matrices in $mub(\mathbb{P}_i)$ has an ICPPL in T_i. If not such partition exists, exit with negative answer.
6: **for** each $1 \leq i \leq r$ **do**
7: Find the ICPPL for $mub(\mathbb{P}_i)$ on T_i
8: **for** each prime submatrix M_x other than $mub(\mathbb{P}_i)$ in $\mathbb{P}_i$ **do**
9: Find ICPIA for M_x
10: **end for**
11: **end for**

3.7 Complexity of Tree Path Assignment – A Discussion

In this section, we briefly make some observations about complexity of TPL and COP. First we present our conclusions on time complexity of TPL and then on space complexity of COP. The latter is a conclusion based on existing literature. However to the best of our knowledge this particular observation about COP's space complexity has not been made earlier.

3.7.1 Time complexity of TPL

Complexity of Compute Feasible Tree Path Labeling for arbitrary trees is in NP. If a solution to Compute Feasible Tree Path Labeling is given, it can be verified using ICPPL and computing bounded (in this case, up to 3-way) intersection cardinalities is polynomial time solvable. Hence the claim. Beyond this, however, we do not have conclusive complexity results for Compute Feasible Tree Path Labeling.

Nonetheless, the problems Compute Interval Labeling and Compute k-subdivided Star Path Labeling are polynomially solvable as we saw in Section 3.5.

Further, we observe that COP testing can in fact be done in logspace. We use two different results in the literature [KKLV10, McC04] to conclude that COP is in logspace. We see this in the next section.

3.7.2 Consecutive Ones Testing is in Logspace

TPL is polynomial time solvable when the given tree is a path. This is the case that is equivalent to COP testing of binary matrices. The known approaches to testing for COP fall into two categories: those that provide a witness when the input matrix does not have COP, and those that do not provide a witness. We mention this aspect of COT in Section 2.3. Section 2.2.3 described generalized PQ-tree where P and Q nodes are called prime and linear nodes. Aside from that, it has a third type of node called degenerate nodes which is present only if the set system does not have COP. Section 2.2.3 further described that using the idea of generalized PQ-tree, [McC04] proves that checking for bipartiteness in the certain incomparability graph is sufficient to check for COP. [McC04] invented a certificate to confirm when a binary matrix does not have COP. Recall [McC04]'s incompatibility graph of a set system $\mathcal{F}$ from Definition 2.3.1 – it has vertices $(a, b), a \neq b$ for every $a, b \in U$, U being the universe of the set system. There are edges $((a, b), (b, c))$ and $((b, a), (c, b))$ if there is a set $S \in \mathcal{F}$ such that $a, c \in S$ and $b \notin S$. In other words the vertices of an edge in this graph represents two orderings that

cannot occur in a consecutive ones ordering of $\mathcal{F}$. In the same section, we also discuss Theorem 2.3.6 which describes the linear certificate for no COP on the incompatibility graph. This is restated below in Theorem 3.7.1.

Theorem 3.7.1 ([McC04, Th. 6.1], [Dom08, Corrected by Th. 2.6, Proof p. 44–47], Theorem 2.3.6). *Let $\mathcal{F}$ be a set family with universe U. Then $\mathcal{F}$ has COP if and only if its incompatibility graph is bipartite and if it does not have the COP, the incompatibility graph has an odd cycle of length at most $n + 3$.*

Thus checking if a set system/hypergraph/binary matrix's incompatibility graph is bipartite is sufficient to conclude the existence of COP. The following result makes this possible in logspace. [Rei84] showed that checking for bipartiteness can be done in logspace. Thus we conclude that consecutive ones testing can be done in logspace.

We also observed logspace complexity of COP using a more recent and independent result – [KKLV10] showed that interval graph isomorphism can be done in logspace. Their paper proves that a canon for interval graphs can be calculated in logspace using an interval hypergraph representation of the interval graph with each hyperedge being a set to which an interval shall be assigned by the canonization algorithm. An overlap graph (subgraph of intersection graph, edges define only strict intersections and no containment) of the hyperedges of the hypergraph is created and canons are computed for each overlap component. The overlap components define a tree like relation due to the fact that two overlap components are such that either all the hyperedges of one is disjoint from all in the other, or all of them are contained in one hyperedge in the other. This is similar to the containment tree defined in [NS09] and in this paper. Finally the canon for the whole graph is created using logspace tree canonization algorithm from [Lin92]. The interval labeling done in this process of canonization is exactly the same as the problem of assigning feasible intervals to a set system, and thus the problem of finding a COP ordering in a binary matrix [NS09].

Theorem 3.7.2 ([KKLV10, Th. 4.7]). *Given an interval hypergraph $\mathcal{H}$, a canonical interval labeling ℓ for $\mathcal{H}$ can be computed in* FL.

We know the equivalence of hypergraphs and set systems (see Definition 2.2.3). Thus interval labeling of hypergraphs obviously translates to finding COP order of a binary matrix which leads to the following corollary.

Corollary 3.7.3. *If a binary matrix M has COP then the interval assignments to each of its columns can be calculated in* FL.

Thus we again conclude that COP testing is indeed in logspace.

CHAPTER 4

CONCLUSION AND FURTHER RESEARCH

In this thesis, we surveyed consecutive-ones property testing algorithms and as new results we gave a characterization for a generalization of COP, namely, feasible tree path labeling of path hypergraphs. The proof for this is constructive and computes a feasibility bijection mapping vertices of the hypergraph to vertices of the given target tree. We also described an exponential algorithm that computes a feasible TPL to a given hypergraph if it is a path hypergraph. In this chapter we will see a few ideas where tree path labeling, in our experience, shows promising scope for further research.

Graph isomorphism and logspace canonization. Canonization is an important tool in graph isomorphism. Invention of a deterministic method of canonization for any class of graphs naturally results in an algorithm for isomorphism. All that is required is to check if the canons of two graphs are the same. Thus one way to study the complexity of graph isomorphism is by studying canonization methods. While general graph isomorphism remains elusive in terms of hardness, canonization has been studied for smaller classes of graphs thus giving complexity or hardness results for them – some of the canonization results are [Lin92, DLN$^+$09, ADKK09, KKLV10].

Ten years ago, [Lin92] made an important discovery that tree isomorphism is in LOGSPACE. It was done by inventing a method of canonization of trees using a logspace depth first traversal algorithm. Less than a couple of years ago, [KKLV10] proved that interval graph canonization is also in LOGSPACE thus drawing logspace conclusions about COT. Interval graphs are FP+C (fixed point logic with counting) definable and [Lau09] showed that this implies that it captures PTIME. This result along with that of undirected graph

reachability being in logspace [Rei08] helped [KKLV10] conclude their logspace result.

How the work in this thesis connects to graph isomorphism and canonization is as follows. For a matrix with the COP, the intersection graph of the input hypergraph is an interval graph. A similar connection exists for TPL to path graphs which is a subclass of chordal graphs and a superclass of interval graphs. Graphs which can be represented as the intersection graph of paths in a tree are path graphs (as seen in Section 1.3). Thus, a hypergraph $\mathcal{F}$ can be labeled with paths in a tree (path labeling as defined in Section 3.2), if and only if the intersection graph $\mathbb{I}(\mathcal{F})$ is a path graph. Path graphs are a subclass of chordal graphs since chordal graphs are characterized as the intersection graphs of subtrees of a tree [Gol04, Ch. 4, Sec. 5]. Chordal graphs are of great significance, are extensively studied, and have several applications. For every chordal graph, there exists a tree and a family of subtrees of this tree such that the intersection graph of this family is isomorphic to the chordal graph [Wal72, Gav74b, Bun74]. These trees when in a certain format, are called *clique trees* [PPY94] of the chordal graph. This is a compact representation of the chordal graph. There has also been work done on the generalization of clique trees to clique hypergraphs [KM02]. Connections between hypergraph isomorphism and properties of set systems is also explored in [BR72] by extending Whitney's theorem of graph isomorphism to hypergraphs. Moreover, [Fou80] generalizes [BR72] and [FG65] by characterizing the isomorphism of two hypergraphs by means of equicardinality of certain edge intersections and the exclusion of certain pairs of subhypergraphs. Thus ICPPL is also a variant of the characterization in [Fou80] with one of the hypergraphs having hyperedges that are paths from a tree (path hypergraph) and ICPPL uses only equicardinality of edge intersections of at most three hyperedges.

Path graphs are well studied in the literature (characterization for UV-graphs was perhaps first mentioned by [Ren70]). Indeed, path graph recognition is a necessary condition for Compute Feasible Tree Path Labeling but not sufficient. We know that path graph recognition can be done in polynomial time [Gav78, first algorithm. $O\left(pn^3\right)$],[Sch93, $O\left(p(m+n)\right)$] [SB94], [DB96, linear algorithm], [Cha11, PR-trees] [1]. Path graph isomorphism is also known be GI-complete [BPT96]. It would be interesting to pursue research to answer the following questions in relation to TPL and graph isomorphism.

1. What does path graph isomorphism hardness tell us about the complexity of Compute Feasible Tree Path Labeling?

2. Can TPL help in computing a canon for path graphs, especially a logspace canon?

Complexity of TPL in other tree classes. We know that on intervals, Compute Feasible Tree Path Labeling is easy (Section 3.5.1). We showed that Compute

1. n is number of vertices, m is number of edges, p is number of maximal cliques in the input graph.

Feasible Tree Path Labeling on k-subdivided star is also polynomial time solvable in Section 3.5.2. However, due to its close connections with path graph isomorphism which is GI-complete [KKLV10], we conjecture that Compute Feasible Tree Path Labeling on general trees cannot do better than GI-complete. Our algorithm in Section 3.6 is unable to solve it in P. It would therefore, be certainly of interest to classify the kinds of trees and hypergraphs for which feasible path labelings can be found efficiently, or better still, to solve Compute Feasible Tree Path Labeling for general trees itself in P-time or prove that it is not in P.

Optimization problems in TPL. There are optimization problems for matrices that do not have COP which we only mention in Section 1.4.2 but did not discuss further since it is not in the scope of the main work of this thesis. We mention them here again to encourage further analogous research in optimization problems of TPL. The central questions in this area are (i) how close is the matrix to having COP, mainly in terms of Tucker's forbidden submatrices, and (ii) how optimally can one alter the matrix to attain COP. With regard to the latter question, recent literature indicate that there has been a lot of interest in matrix modification problems to make a matrix have COP [GH02, TZ07]. While we address the computation of TPL for a given hypergraph from a given target tree in this research, however optimization opportunities in TPL are yet to be explored and their complexity remains open. We suggest a few optimization ideas here and there could be many more such questions that could be asked by a motivated researcher.

1. If Compute Feasible Tree Path Labeling is not possible with the input target tree T, can T be changed to get a feasible TPL?

2. What is the characterization of a hypergraph that will fail the Compute Feasible Tree Path Labeling problem for any target tree? i. e. what is the characterization of a hypergraph that is not a path hypergraph? Can a forbidden substructure be found?

3. If a given hypergraph is not a path hypergraph, can at most k vertices be deleted from the hypergraph to make it a path hypergraph? A dual problem would be to select at least k vertices to induce a path hypergraph? A different flavour of the problem would be deleting (or selecting) hyperedges in the input to get a path hypergraph (analogous to Min COS-R, Min COS-C, Max COS-R, Max COS-C problems in COP).

4. Can at most k vertex memberships in hyperedges be reversed to get a path hypergraph? (analogous to COP Augmentation problem).

It is unlikely that these problems will have efficient exact algorithms since their COP counterparts are all NP hard problems. Thus the scope of research would be to find approximation algorithms or fixed parameter algorithms or hardness results.

x-dimensional generalization of COP. As seen in the chapters of this thesis TPL is a graph theoretic generalization of COP. Intervals are target trees with maximum degree 2 and general trees have no bound on maximum degree. Another way to look at intervals is using geometry. Intervals can be imagined in one dimension as line segments along a single line in the Euclidean space and the elements of hypergraph universe are coordinate points on this line segment. The generalization of this visualization is obvious – how does the interval labeling property generalize when each hyperedge must be mapped to coordinate points bounded by a rectilinear 2-D shape on a single plane, say rectangles (analogous to coordinate points on line segments in interval labeling problem). One hardness result is already known – the intersection graphs of axis-parallel boxes in $\mathbb{R}^d$ is GI-complete [Ueh08].

BIBLIOGRAPHY

[ABH98] J. E. Atkins, E. G. Boman, and B. Hendrickson. A spectral algorithm for seriation and the consecutive ones problem. *SICOMP: SIAM Journal on Computing*, 28, 1998.

[ADKK09] V. Arvind, Bireswar Das, Johannes K´obler, and Sebastian Kuhnert. The isomorphism problem for k-trees is complete for logspace. *Electronic Colloquium on Computational Complexity*, –TBD–(53), 2009.

[ADP80] Giorgio Ausiello, Alessandro D'Atri, and Marco Protasi. Structure preserving reductions among convex optimization problems. *J. Comput. Syst. Sci*, 21(1):136–153, 1980.

[ALC67] S. Even A. Lempel and I. Cederbaum. An algorithm for planarity testing of graphs. *Theory of Graphs*, pages 215–232, 1967.

[AS95] Annexstein and Swaminathan. On testing consecutive-ones property in parallel. In *SPAA: Annual ACM Symposium on Parallel Algorithms and Architectures*, 1995.

[Ben59] S. Benzer. On the topology of the genetic fine structure. *Proc. Nat. Acad. Sci. U.S.A.*, 45:1607–1620, 1959.

[BJHY00] Jorgen Bang-Jensen, Jing Huang, and Anders Yeo. Convex-round and concave-round graphs. *SIAM Journal on Discrete Mathematics*, 13(2):179–193, 2000.

[BL76] Kellogg S. Booth and George S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, December 1976.

[BLS99] Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[Boo75] Kellogg S. Booth. *PQ-tree algorithms*. PhD thesis, Univ. California, Berkeley, 1975.

[BP92] J. R. S. Blair and B. Peyton. An introduction to chordal graphs and clique trees. Technical report, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, 1992.

[BPT96] L. Babel, I. N. Ponomarenko, and G. Tinhofer. The isomorphism problem for directed path graphs and for rooted directed path graphs. *J. Algorithms*, 21(3):542–564, 1996.

[BR72] C Berge and R Rado. Note on isomorphic hypergraphs and some extensions of whitney's theorem to families of sets. *Journal of Combinatorial Theory, Series B*, 13(3):226 – 241, 1972.

[BS03] David A. Bader and Sukanya Sreshta. A new parallel algorithm for planarity testing, April 11 2003.

[BTV09] Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Trans. Comput. Theory*, 1:4:1–4:17, February 2009.

[Bun74] Peter Buneman. A characterization of rigid circuit graphs. *Discrete Math.*, 9:205–212, 1974.

[Cha11] Steven Chaplick. Characterizing and recognizing path graphs and directed path graphs using PR-trees. *Discrete Applied Mathematics (submitted)*, March 2011. Preprint submitted. Revision available at http://www.cs.toronto.edu/ chaplick/PRtrees.pdf.

[CKL96] R. Chandrasekaran, S. N. Kabadi, and S. Lakshminarayanan. An extension of a theorem of Fulkerson and Gross. *Linear Algebra and its Applications*, 246(1–3):23–29, October 1996.

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw Hill, Boston, MA, USA, 2001.

[COR98] Thomas Christof, Marcus Oswald, and Gerhard Reinelt. Consecutive ones and a betweenness problem in computational biology. *Lecture Notes in Computer Science*, 1412, 1998.

[CY91] Lin Chen and Yaacov Yesha. Parallel recognition of the consecutive ones property with applications. *J. Algorithms*, 12(3):375–392, 1991.

[CY95] Gen-Huey Chen and Chang-Wu Yu. Efficient parallel algorithms for doubly convex-bipartite graphs. *TCS: Theoretical Computer Science*, 147:249–265, 1995.

[DB96] E. Dahlhaus and G. Bailey. Recognition of path graphs in linear time. In *The Fifth Italian Conference on Theoretical Computer Science (Revello, 1995)*, pages 201–210, River Edge, NJ, 1996. World Sci. Publishing.

[DF99] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[DFH+06] Dujmovic, Fellows, Hallett, Kitching, Liotta, McCartin, Nishimura, Ragde, Rosamond, Suderman, Whitesides, and Wood. A fixed-parameter approach to 2-layer planarization. *ALGRTHMICA: Algorithmica*, 45, 2006.

[DGN07] Michael Dom, Jiong Guo, and Rolf Niedermeier. Approximability and parameterized complexity of consecutive ones submatrix problems. In *Theory and Applications of Models of Computation, 4th International Conference, TAMC 2007, Shanghai, China, May 22-25, 2007, Proceedings*, volume 4484 of *Lecture Notes in Computer Science*, pages 680–691. Springer, 2007.

[DLN+09] Samir Datta, Nutan Limaye, Prajakta Nimbhorkar, Thomas Thierauf, and Fabian Wagner. Planar graph isomorphism is in log-space. – *TTBBDD – Algebraic Methods in Computational Complexity*, 2009.

[Dom08] Michael Dom. *Recognition, Generation, and Application of Binary Matrices with the Consecutive-Ones Property*. PhD thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany, 2008. Published by Cuvillier, 2009.

[Fei98] Uriel Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[Fer05a] H. Fernau. *Parameterized Algorithmics: A Graph-Theoretic Approach*. Habilitationsschrift, Universität Tübingen, Germany, 2005.

[Fer05b] H. Fernau. Two-layer planarization: improving on parameterized algorithmics. In *SOFSEM*, volume 3381 of *LNCS*, pages 137–146. Springer, 2005.

[Fer08] Henning Fernau. Parameterized algorithmics for linear arrangement problems. *Discrete Appl. Math.*, 156:3166–3177, October 2008.

[FG65] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pac. J. Math.*, 15:835–855, 1965.

[Fis85] Peter C. Fishburn. *Interval Orders and Interval Graphs*. Wiley, 1985.

[Fou80] J.-C. Fournier. Isomorphismes d'hypergraphes par intersections équicardinales d'arêtes et configurations exclues. *J. Comb. Theory, Ser. B*, 29(3):321–327, 1980.

[Gar07] Frederic Gardi. The roberts characterization of proper and unit interval graphs. *Discrete Mathematics*, 307(22):2906–2908, 2007.

[Gav74a] Fanica Gavril. Algorithms on circular-arc graphs. *Networks*, 4:357–369, 1974.

[Gav74b] Fanica Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *J. Combin. Theory*, 16:47–56, 1974.

[Gav78] Fanica Gavril. A recognition algorithm for the intersection graphs of paths in trees. *Discrete Mathematics*, 23(3):211 – 227, 1978.

[GH64] P. C. Gilmore and A. J. Hoffman. A characterization of comparability graphs and of interval graphs. *Can. J. Math.*, 16:539–548, 1964.

[GH02] Ganjali and Hajiaghayi. A note on the consecutive ones submatrix problem. *IPL: Information Processing Letters*, 83, 2002.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability.* Freeman, San Francisco, 1979.

[GM03] Marisa Gutierrez and João Meidanis. Recognizing clique graphs of directed edge path graphs. *Discrete Applied Mathematics*, 126:297–304, 2003.

[Gol04] Martin Charles Golumbic. *Algorithmic graph theory and perfect graphs*, volume 57 of *Annals of Discrete Mathematics.* Elsevier Science B.V., 2004. Second Edition.

[HL06] Dorit S. Hochbaum and Asaf Levin. Cyclical scheduling and multi-shift scheduling: Complexity and approximation algorithms. *Discrete Optimization*, 3(4):327–340, 2006.

[HM03] Wen-Lian Hsu and Ross M. McConnell. PC trees and circular-ones arrangements. *Theoretical Computer Science*, 296:99–116, 2003.

[HMPV00] Michel Habib, Ross M. McConnell, Christophe Paul, and Laurent Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1–2):59–84, 2000.

[Hsu92] Wen-Lian Hsu. A simple test for the consecutive ones property. In *Proc. of the ISAAC'92* [Hsu02], pages 459–469. Later appeared in J. Algorithms 2002.

[Hsu01] Wen-Lian Hsu. PC-trees vs. PQ-trees. *Lecture Notes in Computer Science*, 2108:207–217, 2001.

[Hsu02] Wen-Lian Hsu. A simple test for the consecutive ones property. *J. Algorithms*, 43(1):1–16, 2002.

[HT98] T. C. Hu and P. A. Tucker. Minimax programs, bitonic columns and PQ trees, November 29 1998.

[HT02] Hochbaum and Tucker. Minimax problems with bitonic matrices. *Networks: An International Journal*, 40, 2002.

[JKC+04] Johnson, Krishnan, Chhugani, Kumar, and Venkatasubramanian. Compressing large boolean matrices using reordering techniques. In *VLDB: International Conference on Very Large Data Bases.* Morgan Kaufmann Publishers, 2004.

[JLL76] Neil D. Jones, Y. Edmund Lien, and William T. Laaser. New problems complete for nondeterministic log space. *MST: Mathematical Systems Theory*, 10, 1976.

[Ken69] D. Kendall. Incidence matrices, interval graphs and seriation in archaeology. *Pacific Journal of Mathematics*, 1969.

[KKLV10] Johannes Köbler, Sebastian Kuhnert, Bastian Laubner, and Oleg Verbitsky. Interval graphs: Canonical representation in logspace. *Electronic Colloquium on Computational Complexity (ECCC)*, 17:43, 2010.

[KM89] N. Korte and R.H. Moehring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.*, pages 68–81, 1989.

[KM02] P. S. Kumar and C. E. Veni Madhavan. Clique tree generalization and new subclasses of chordal graphs. *Discrete Applied Mathematics*, 117:109–131, 2002.

[Kou77] Lawrence T. Kou. Polynomial complete consecutive information retrieval problems. *SIAM Journal on Computing*, 6(1):67–75, March 1977.

[KR88] P.N. Klein and J.H. Reif. An efficient parallel algorithm for planarity. *Journal of Computer and System Science*, 18:190–246, 1988.

[Lau09] Bastian Laubner. Capturing polynomial time on interval graphs. *CoRR*, abs/0911.3799, 2009. informal publication.

[Lau10] Bastian Laubner. Capturing polynomial time on interval graphs. In *LICS*, pages 199–208. IEEE Computer Society, 2010.

[LB63] C. G. Lekkerkerker and J. Ch. Boland. Representation of a finite graph by a set of intervals on the real line. *Fundamenta Mathematicae*, 51:45–64, 1962/1963.

[Lin92] Steven Lindell. A logspace algorithm for tree canonization (extended abstract). In *STOC*, pages 400–404. ACM, 1992.

[McC04] Ross M. McConnell. A certifying algorithm for the consecutive-ones property. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 2004.

[MM95] J. Meidanis and Erasmo G. Munuera. A simple linear time algorithm for binary phylogeny. In N. Ziviani, J. Piquer, B. Ribeiro, and R. Baeza-Yates, editors, *Proc. of the XV International Conference of the Chilean Computing Society*, pages 275–283, Nov 1995.

[MM96] J. Meidanis and E. G. Munuera. A theory for the consecutive ones property. In *Proc. of the III South American Workshop on String Processing* [MPT98], pages 194–202.

[MPT98] Meidanis, Porto, and Telles. On the consecutive ones property. *DAMATH: Discrete Applied Mathematics and Combinatorial Operations Research and Computer Science*, 88, 1998.

[MW86] C.L. Monma and V.K. Wei. Intersection graphs of paths in a tree. *J. Combin. Theory Ser. B*, (41):141–181, 1986.

[Nov89] Mark B. Novick. Generalized PQ-trees. Technical Report 1074, Dept. of Computer Science, Cornell University, Ithaca, NY 14853-7501, Dec 1989.

[NS09] N. S. Narayanaswamy and R. Subashini. A new characterization of matrices with the consecutive ones property. *Discrete Applied Mathematics*, 157(18):3721–3727, 2009.

[PPY94] Barry W. Peyton, Alex Pothen, and Xiaoqing Yuan. A clique tree algorithm for partitioning a chordal graph into transitive subgraphs. Technical report, Old Dominion University, Norfolk, VA, USA, 1994.

[Rei84] John H. Reif. Symmetric complementation. *JACM: Journal of the ACM*, 31(2):401–421, 1984.

[Rei08] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

[Ren70] Peter L. Renz. Intersection representations of graphs by arcs. *Pacific J. Math.*, 34(2):501–510, 1970.

[Rob51] W. S. Robinson. A method for chronologically ordering archaeological deposits. *American Antiquity*, 16(4):293–301, 1951.

[Rob69] Fred. S. Roberts. Indierence graphs. In Frank Harary, editor, *Proof Techniques in Graph Theory*, pages 139–146. Academic Press, 1969.

[SB94] J.L. Szwarcfiter and C.F. Bornstein. Clique graphs of chordal and path graphs. *SIAM. J. Disc. Math.*, 7:331–336, 1994.

[Sch93] Alejandro A. Schaffer. A faster algorithm to recognize undirected path graphs. *Discrete Applied Mathematics*, 43:261–295, 1993.

[SH99] W.K. Shih and W.L. Hsu. Note a new planarity test. *TCS: Theoretical Computer Science*, 223:179–191, 1999.

[SW05] M. Suderman and S. Whitesides. Experiments with the fixed-parameter approach for two-layer planarization. *—-jgaa—-*, 9(1):149–163, 2005.

[TM05] Guilherme P. Telles and João Meidanis. Building PQR trees in almost-linear time. *Electronic Notes in Discrete Mathematics*, 19:33–39, 2005.

[Tuc72] Alan Tucker. A structure theorem for the consecutive 1's property. *J. Comb. Theory Series B*, 12:153–162, 1972.

[TZ04] Jinsong Tan and Louxin Zhang. Approximation algorithms for the consecutive ones submatrix problem on sparse matrices. volume 3341 of *Lecture Notes in Computer Science*, pages 835–846, 2004.

[TZ07] J. Tan and L. Zhang. The consecutive ones submatrix problem for sparse matrices. *Algorithmica*, 48(3):287–299, 2007.

[Ueh08] R. Uehara. Simple geometrical intersection graphs. In *WALCOM*, volume 4921 of *LNCS*, pages 25–33. Springer, 2008.

[Vel82] Marinus Veldhorst. An analysis of sparse matrix storage schemes. In *Mathematical Centre Tract (Vol. 150)*, Mathematical Centre, Amsterdam, the Netherlands, 1982.

[Vel85] Marinus Veldhorst. Approximation of the consecutive ones matrix augmentation problem. *SIAM Journal on Computing*, 14(3):709–729, August 1985.

[Wal72] J. R. Walter. *Representation of rigid cycle graphs.* PhD thesis, Wayne State University, 1972.

[Wal96] David Foster Wallace. The string theory. *Esquire*, 126(1):56(14), July 1996.

[Wal99] David Foster Wallace. *Brief Interviews with Hideous Men.* Little, Brown and Company, May 1999.

[Wal00] David Foster Wallace. Rhetoric and the math melodrama. *Science*, 290(22):2263–2267, December 2000.

[Wal10] David Foster Wallace. *Fate, Time, and Language: An Essay on Free Will.* Columbia University Press, reprint edition, December 2010.