# 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**

**JANUARY 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
10 January 2012

# ACKNOWLEDGEMENTS

# ABSTRACT

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

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.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABBREVIATIONS

| | |
|---|---|
| **COP** | Consecutive-ones Property |
| **COT** | Consecutive-ones Testing |
| **ICPIA** | Intersection Cardinality Preservation Interval Assignment |
| **ICPPL** | Intersection Cardinality Preserved Path Labeling |

# NOTATION

| | |
|---|---|
| $2^U$ | Powerset of set $U$ |

# 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.

Section 1.3 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.

## 1.1 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"*, *"The String $\mathbb{T}$heory"*, *"[$\mathbb{W}$]Rhetoric and the Math Melodrama"* and *"$\mathbb{F}$ate, Time, and Language: An Essay on Free Will"*. 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.

$$\begin{array}{rcl}
U & = & \{\mathbf{Pa},\ \mathbf{Pi},\ \mathbf{Sn},\ \mathbf{Wo},\ \mathbf{Vi},\ \mathbf{Li},\ \mathbf{Ch},\ \mathbf{Sa},\ \mathbf{Fr},\ \mathbf{Sc},\ \mathbf{Lu}\} \\
\mathcal{F} & = & \{\mathbb{B}, \mathbb{T}, \mathbb{W}, \mathbb{F}\} \\
\mathbb{B} & = & \{\mathbf{Ch},\ \mathbf{Sa},\ \mathbf{Fr},\ \mathbf{Sc},\ \mathbf{Lu}\} \\
\mathbb{T} & = & \{\mathbf{Pa},\ \mathbf{Pi},\ \mathbf{Vi},\ \mathbf{Ch}\} \\
\mathbb{W} & = & \{\mathbf{Sn},\ \mathbf{Pi},\ \mathbf{Wo}\} \\
\mathbb{F} & = & \{\mathbf{Vi},\ \mathbf{Li},\ \mathbf{Ch},\ \mathbf{Fr}\} \\
n & = & |U| = 11 \\
m & = & |\mathcal{F}| = 4
\end{array}$$

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

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. Fig 1.2 shows the street map for *Infinite Loop*. It may be noted that as a graph, it classifies as a tree.

$$\begin{array}{rcl}
T & = & \textit{Street map tree of } \text{Infinite Loop} \\
V(T) & = & \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\} \\
\mathcal{P} & = & \{R\mathbb{B}, R\mathbb{T}, R\mathbb{W}, R\mathbb{F}\} \\
R\mathbb{B} & = & \{9, 1, 5, 3, 11\} \\
R\mathbb{T} & = & \{7, 2, 6, 5\} \\
R\mathbb{W} & = & \{8, 2, 4\} \\
R\mathbb{F} & = & \{10, 6, 5, 3\} \\
n & = & |V| = 11 \\
m & = & |\mathcal{P}| = 4 \\
 & & \\
\ell & = & \textit{Study group to route mapping} \\
\ell(\mathbb{X}) & = & R\mathbb{X} \text{ for all } \mathbb{X} \in \mathcal{F}
\end{array}$$

| *Apartment allocation* ($\phi$) | |
|---|---|
| 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

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 specifically, we are interested in enforcing 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 Fig 1.2 and the students' allocation of apartments that obeys this route allocation is also shown. Table 1.2 shows the same solution set theoretically. How this is algorithmically computed is the focus of this thesis.

As a special case, 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. It is debatable if this will not hamper the attention span and memory retention rate of the students but that is, regrettably, out of the scope of this thesis. 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-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

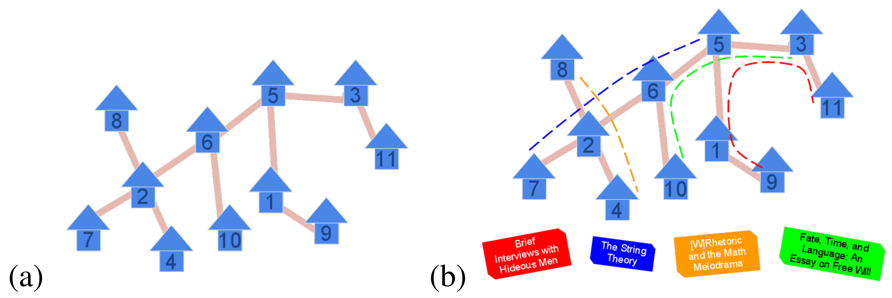

(a)
8
6
5
3
11
2
1
9
7
4
10
(b)
Brief Interviews with Hideous Men
The String Theory
[W]Rhetoric and the Math Melodrama
Fate, Time, and Language: An Essay on Free Will

Figure 1.1: (a) *Infinite Loop* street map (b) *Infinite Loop* street map with study group routes allocated. Routes are color coded as follows: red for $\mathbb{B}$ group, blue for $\mathbb{T}$ group, orange for $\mathbb{W}$ group, green for $\mathbb{F}$ group

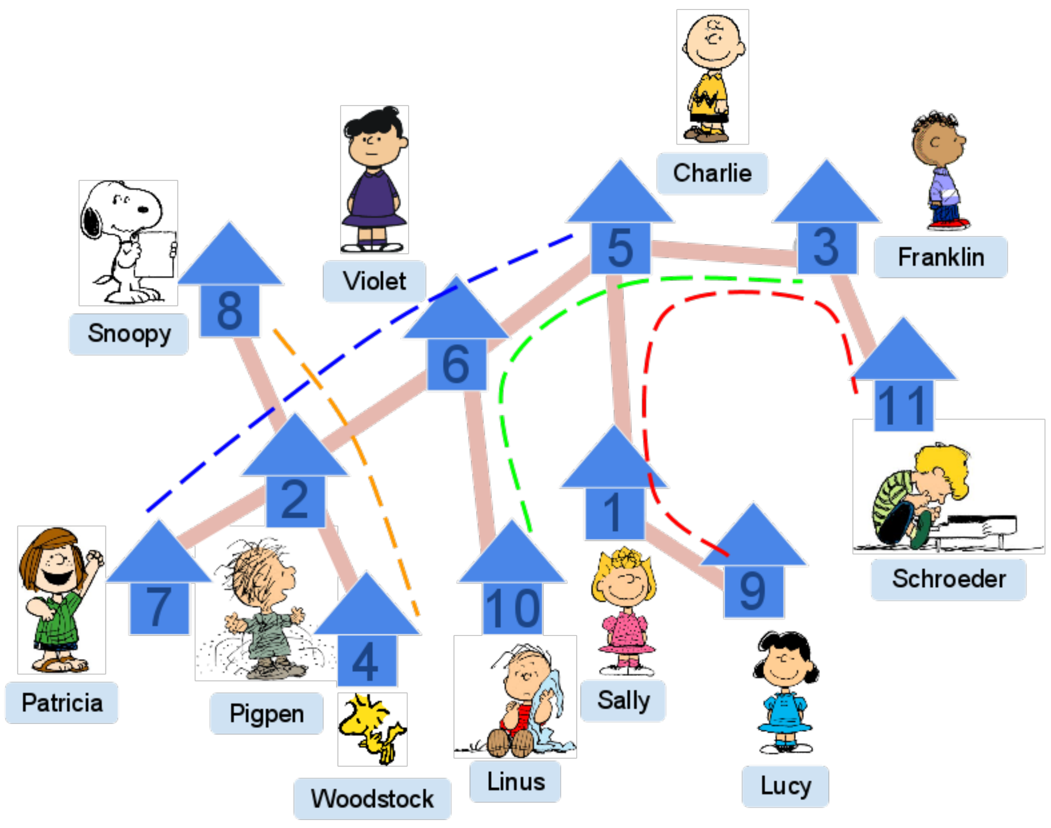


Figure 1.2: Individual allocation of apartments to students in *Infinite Loop* that meets the requirements stated before. The routes are color coded as follows: red for $\mathbb{B}$ group, blue for $\mathbb{T}$ group, orange for $\mathbb{W}$ group, green for $\mathbb{F}$ group. *Peanuts images © Charles Schulz*

$$M_1: \begin{array}{cccc} c_1 & c_2 & c_3 & c_4 \\ \mathbf{1} & 0 & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & \mathbf{1} \\ \mathbf{1} & 0 & 0 & \mathbf{1} \end{array} \quad \Bigg| \quad M_1': \begin{array}{cccc} c_3 & c_1 & c_4 & c_2 \\ \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} \\ 0 & \mathbf{1} & \mathbf{1} & 0 \end{array} \qquad M_2: \begin{array}{cccc} d_1 & d_2 & d_3 & d_4 \\ \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & 0 \\ 0 & \mathbf{1} & 0 & \mathbf{1} \end{array}$$

Figure 1.3: Matrices with and without COP. $M_1$ has COP because by permuting its columns, $c_1$-$c_4$, one can obtain $M_1'$ where the **1**s in each row are consecutive. $M_2$, however, does not have COP since no permutation of its columns, $d_1$-$d_4$, will arrange **1**s in each row consecutively [Dom08].

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).

## 1.2 Basic preliminaries - general definitions and nomenclature

## 1.3 Consecutive-ones Property Testing - a Brief Survey

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

### 1.3.1 Matrices with COP

As seen earlier, the interval assignment problem (illustrated as the course scheduling problem in Section 1.1), 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). Figure 1.3 shows examples of this property. 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 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 trees 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 of matrix 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].

### 1.3.2 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. 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 if and only if the matrix has COP 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 has at most $\Delta$ **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. [HG02] 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. As mentioned earlier, this problem is known to be NP complete due to [Boo75]. [Vel85] also 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.

## 1.4 ********* Application of COP in Areas of Graph Theory and Algorithms

### 1.4.1 ********* COP in Relational Database Model

### 1.4.2 ********* COP in Graph Isomorphism

### 1.4.3 ********* Certifying Algorithms

## 1.5 Generalization of COP - the Motivation

Section 1.3.1 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 statisfaction 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 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[1] 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 paper) successfully. 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[2] is feasible.

[1]The notion of *feasibility* is formally defined in Section 3.2.

[2]The terms *tree path labeling* and *tree path assignment* are, in informal language, synonyms. Formally, the former refers to the bijection $\ell : \mathcal{F} \rightarrow \mathcal{P}$. 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 $T$.

## 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[3] 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 [4] (TPL) feasible, which is a generalization of the COP problem. This problem is defined as follows.

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 $\phi$ when applied on any hyperedge in $\mathcal{F}$ will give the path mapped to it by the given tree path labeling $\ell$. i.e., $\ell(S) = \{\phi(x) \mid x \in S\}$, for every hyperedge $S \in \mathcal{F}$. |

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[5]. 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 com-

[3]A *hypergraph* is an alternate representation of a set system and will be used in this thesis. See Section 3.2 for the formal definition.

[4]A *tree path labeling* $\ell$ 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.

[5]See Section 3.3 for the definition of ICPPL.

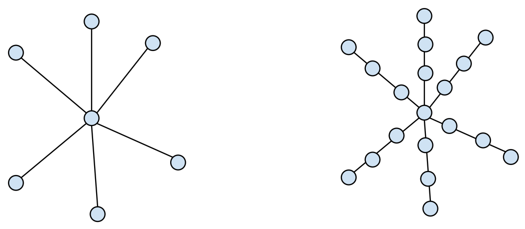

(a) (b)

Figure 1.4: Examples of $k$-subdivided stars. (a) $k = 0$ (b) $k = 2$

puting a feasible TPL for a given hypergraph and target tree, if one exists. This problem, COMPUTE FEASIBLE TREE PATH LABELING, is defined as follows.

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 $\ell : \mathcal{F} \rightarrow \mathcal{P}$, such that FEASIBLE TREE PATH LABELING returns **true** on $(\mathcal{F}, T, \ell)$. |

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* and when the hyperedges in the hypergraph $\mathcal{F}$ have at most $k + 2$ vertices. A couple of examples of $k$-subdivided stars are given in Figure 1.4.

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 $\ell : \mathcal{F} \rightarrow \mathcal{P}$, such that FEASIBLE TREE PATH LABELING returns **true** on $(\mathcal{F}, T, \ell)$. |

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[6] are independent except for the root[7] 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 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 size 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*[8] relation between the sets involved is an obvious equivalence relation, $\mathcal{F}$ quite naturally partitions into equivalence classes known as *overlap components*. In the context of COP, overlap components were used in [Hsu02] and [KKLV10]. Moreover, [NS09] discovered that these equivalence classes form a total order . 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.

[6]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.

[7]The vertex with maximum degree in a $k$-subdivided star is called *root*. See Section 3.2.

[8]

[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

## 2.1 Matrices with COP

## 2.2 Optimization problems in COP

## 2.3 ********* COP in Relational Database Model

### 2.3.1 ********* COP in Graph Isomorphism

### 2.3.2 ********* Certifying Algorithms

# CHAPTER 3

# Tree Path Labeling of Path Hypergraphs - the 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 parent problem addressed in this thesis. 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.

The necessary preliminaries with definitions etc. are presented in Section 3.2. Section 3.3 documents the characterization of a feasible path labeling for any path labeling from any target trees. Section 3.4 describes the special case where the target tree is of a particular family of trees and specifically, Section 3.4.2 describes a polynomial time algorithm to find the tree path labeling of a given set system from a given $k$-subdivided tree. Section 3.5 describes the general situation where the target tree has no restrictions and the algorithm to find a TPL, if any, in this case.

## 3.1 Introduction

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 [NS09] 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 its interval. The result in [NS09] characterizes interval assignments to the sets which can be obtained from a single permutation of the rows (columns). They show that for each set, 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. . Finally, the idea of decomposing a given binary matrix into prime matrices to check for COP is adopted from cite|wlh02| to test if an ICPIA exists for a given set system.

A natural generalization of the interval assignment problem is feasible tree path labeling problem of a set system. The problem is defined as follows – given a set system $\mathcal{F}$ from a universe $U$ and a 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 *tree path labeling problem* for an input set system, target tree pair – $(\mathcal{F}, T)$. 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 $\phi$ when applied on any hyperedge in $\mathcal{F}$ will give the path mapped to it by the given tree path labeling $\ell$. i.e., $\ell(S) = \{\phi(x) \mid x \in S\}$, for every hyperedge $S \in \mathcal{F}$. |

As a special case if the tree $T$ is a path, the problem becomes the interval assignment problem. We focus on the question of 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 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 and the input runs a filtering algorithm (described in this paper) successfully. This characterization is proved constructively and it gives a natural data structure that stores all the relevant feasible bijections between $U$ and $V(T)$. Further, the filtering algorithm is also an efficient algorithm to test if a tree path labeling to the set system is feasible. This generalizes the result in [NS09].

In the later part of this paper, we focus on a new special case of the tree path labeling problem. Here the set system is such that every set is at most $k+2$ in size and for every pair of sets in it there exists a sequence of sets between them with consecutive sets in this sequence having a strict intersection – i.e., non-empty intersection with neither being contained in the other. Moreover, the given tree is a *$k$-subdivided star*. We demonstrate a polynomial time algorithm to find a feasible path labeling in this case.

## 3.2 Preliminaries to new results

This section states definitions and basic facts necessary in the scope of this document.

The set $\mathcal{F} \subseteq (2^U \setminus \emptyset)$ is a *set system* of a universe $U$ with $|U| = n$. The *support* of a set system $\mathcal{F}$ denoted by $supp(\mathcal{F})$ is the union of all the sets in $\mathcal{F}$; $supp(\mathcal{F}) = \bigcup_{S \in \mathcal{F}} S$. For the purposes of this paper, a set system is required to "cover" the universe; $supp(\mathcal{F}) = U$.

The graph $T$ represents a *target tree* with same number of vertices as elements in $U$; $|V(T)| = n$. 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 set system $\mathcal{F}$ can be alternatively represented by a *hypergraph* $\mathcal{F}_H$ whose vertex set is $supp(\mathcal{F})$ 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 in the scope of this paper, we drop the subscript $_H$ in the notation and refer to both the structures by $\mathcal{F}$.

Two hypergraphs $\mathcal{F}'$, $\mathcal{F}''$ are said to be *isomorphic* to each other, denoted by $\mathcal{F}' \cong \mathcal{F}''$, iff there exists a bijection $\phi : supp(\mathcal{F}') \rightarrow supp(\mathcal{F}'')$ such that for all sets $A \subseteq supp(\mathcal{F}')$, $A$ is a hyperedge in $\mathcal{F}'$ iff $B$ is a hyperedge in $\mathcal{F}''$ where $B = \{\phi(x) \mid x \in A\}$ [KKLV10]. This is called *hypergraph isomorphism*.

The *intersection graph* $\mathbb{I}(\mathcal{F})$ 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 vertices iff their corresponding hyperedges have a non-empty intersection [Gol04].

If the intersection graphs of two hypergraphs are isomorphic, $\mathbb{I}(\mathcal{F}) \cong \mathbb{I}(\mathcal{P})$ where $\mathcal{P}$ is also a path system, then the bijection $\ell : \mathcal{F} \rightarrow \mathcal{P}$ due to this isomorphism is called a *path labeling* of the hypergraph $\mathcal{F}$. To illustrate further, let $g : V(\mathcal{F}) \rightarrow V(\mathcal{P})$ be the above mentioned isomorphism where $V(\mathcal{F})$ and $V(\mathcal{P})$ are the vertex sets that represent the hyperedges for each hypergraph respectively, $V(\mathcal{F}) = \{v_S \mid S \in \mathcal{F}\}$ and $V(\mathcal{P}) = \{v_P \mid P \in \mathcal{P}\}$. Then the path labeling $\ell$ is defined as follows: $\ell(S_1) = P_1$ iff $g(v_{S_1}) = v_{P_1}$. The path system $\mathcal{P}$ may be alternatively denoted in terms of $\mathcal{F}$ and $\ell$ as $\mathcal{F}^\ell$. In most scenarios in this paper, what is given are the pair $(\mathcal{F}, \ell)$ and the target tree $T$; hence this notation will be used more often.

If $\mathcal{F} \cong \mathcal{P}$ where $\mathcal{P}$ is a path system, then $\mathcal{F}$ is called a *path hypergraph* and $\mathcal{P}$ is called *path representation* of $\mathcal{F}$. If this isomorphism is $\phi : supp(\mathcal{F}) \rightarrow V(T)$, then it is clear that there is an *induced path labeling* $\ell_\phi : \mathcal{F} \rightarrow \mathcal{P}$ to the set system; $\ell_\phi(S) = \{y \mid y = \phi(x), x \in S\}$ for all $S \in \mathcal{F}$. Recall that $supp(\mathcal{P}) = V(T)$.

A path labeling $(\mathcal{F}, \ell)$ is defined to be *feasible* if $\mathcal{F} \cong \mathcal{F}^\ell$ and this hypergraph isomorphism $\phi : supp(\mathcal{F}) \rightarrow supp(\mathcal{F}^\ell)$ induces a path labeling $\ell_\phi : \mathcal{F} \rightarrow \mathcal{F}^\ell$ such that $\ell_\phi = \ell$.

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 iff their corresponding hyperedges overlap. 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; $S \between S'$ iff $S \cap S' \neq \emptyset, S \nsubseteq S', S' \nsubseteq S$. 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]. Additionally, if $S$ is such that it is contained in no other hyperedge in $\mathcal{F}$, i.e., it is inclusion maximal then it is called *super-marginal*.

A *star* graph is a complete bipartite graph $K_{1,p}$ which is clearly a tree and $p$ is the number of leaves. The vertex with maximum degree is called the *center* of the star and the edges are called *rays* of the star. A *$k$-subdivided star* is a star with all its rays subdivided exactly $k$ times. The definition of a *ray of a $k$-subdivided star* is extended to the path from the center to a leaf. It is clear that all rays are of length $k+2$.

## 3.3 Characterization of Feasible Tree Path Labeling

In this section we give an algorithmic characterization of a feasibility of tree path labeling. 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.

| | | |
|---|---|---|
| (Property i) | $\lvert S\rvert = \lvert \ell(S)\rvert$ | for all $S \in \mathcal{F}$ |
| (Property ii) | $\lvert S_1 \cap S_2\rvert = \lvert \ell(S_1) \cap \ell(S_2)\rvert$ | for all distinct $S_1, S_2 \in \mathcal{F}$ |
| (Property iii) | $\lvert S_1 \cap S_2 \cap S_3\rvert = \lvert \ell(S_1) \cap \ell(S_2) \cap \ell(S_3)\rvert$ | for all distinct $S_1, S_2, S_3 \in \mathcal{F}$ |

The following lemma is useful in subsequent arguments.

**Lemma 3.3.1** *If $\ell$ 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, $|S_1 \cap S_2| - |S_1 \cap S_2 \cap S_3| = |P_1 \cap P_2| - |P_1 \cap P_2 \cap P_3| = |(P_1 \cap P_2) \setminus P_3| = |P_1 \cap (P_2 \setminus P_3)|$. Thus lemma is proven. ∎

In the remaining part of this section we show that $(\mathcal{F}, \ell)$ is feasible if and only if it is an ICPPL and Algorithm 3 returns a non-empty function. Algorithm 3 recursively does two levels of filtering of $(\mathcal{F}, \ell)$ to make it simpler while retaining the set of isomorphisms, if any, between $\mathcal{F}$ and $\mathcal{F}^\ell$. First, we present Algorithm 5 or `filter_1`, 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_1` $(\mathcal{F}, \ell, T)$

```
1: $\mathcal{F}_0 \leftarrow \mathcal{F}$, $\ell_0(S) \leftarrow \ell(S)$ for all $S \in \mathcal{F}_0$
2: $j \leftarrow 1$
3: 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
4:     $\mathcal{F}_j \leftarrow (\mathcal{F}_{j-1} \setminus \{S_1, S_2\}) \cup \{S_1 \cap S_2, S_1 \setminus S_2, S_2 \setminus S_1\}$
       | Remove $S_1$, $S_2$ and add the "filtered" sets
5:     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
       | Carry forward the path labeling for all existing sets other than $S_1$, $S_2$
6:     $\ell_j(S_1 \cap S_2) \leftarrow \ell_{j-1}(S_1) \cap \ell_{j-1}(S_2)$
7:     $\ell_j(S_1 \setminus S_2) \leftarrow \ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2)$
       | Define path labeling for new sets
8:     $\ell_j(S_2 \setminus S_1) \leftarrow \ell_{j-1}(S_2) \setminus \ell_{j-1}(S_1)$
9:     if $(\mathcal{F}_j, \ell_j)$ does not satisfy (Property iii) of ICPPL then
10:        exit
11:    end if
12:    $j \leftarrow j + 1$
13: end while
14: $\mathcal{F}' \leftarrow \mathcal{F}_j$, $\ell' \leftarrow \ell_j$
15: return $(\mathcal{F}', \ell')$
```

**Lemma 3.3.2** *In Algorithm 5, if input $(\mathcal{F}, \ell)$ is a feasible path assignment then at the end of $j$th iteration of the* **while** *loop, $j \geq 0$, $(\mathcal{F}_j, \ell_j)$ is a feasible path assignment.*

**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 $\phi : supp\,(\mathcal{F}_{j-1}) \rightarrow V\,(T)$ that defines $(\mathcal{F}, \ell)$'s feasibility, is such that the induced path labeling on $\mathcal{F}_{j-1}$, $\ell_{\phi[\mathcal{F}_{j-1}]}$ is equal to $\ell_{j-1}$. We will prove that $\phi$ 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 $\mathcal{F}_{j-1}$. For the same reason and $\phi$ being a bijection, it is clear that when applying the $\phi$ induced path labeling on $\mathcal{F}_j$, $\ell_{\phi[\mathcal{F}_j]}(S_1 \setminus S_2) = \ell_{\phi[\mathcal{F}_{j-1}]}(S_1) \setminus \ell_{\phi[\mathcal{F}_{j-1}]}(S_2)$. Now observe that $\ell_j(S_1 \setminus S_2) = \ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2) = \ell_{\phi[\mathcal{F}_{j-1}]}(S_1) \setminus \ell_{\phi[\mathcal{F}_{j-1}]}(S_2)$. Thus the induced path labeling $\ell_{\phi[\mathcal{F}_j]} = \ell_j$. Therefore lemma is proven. ∎

**Lemma 3.3.3** *In Algorithm 5, at the end of $j$th iteration, $j \geq 0$, of the* **while** *loop, the following invariants are maintained.*

*I* $\quad \ell_j(R)$ *is a path in $T$,* $\qquad$ *for all $R \in \mathcal{F}_j$*

*II* $|R| = |\ell_j(R)|$, *for all* $R \in \mathcal{F}_j$

*III* $|R \cap R'| = |\ell_j(R) \cap \ell_j(R')|$, *for all* $R, R' \in \mathcal{F}_j$

*IV* $|R \cap R' \cap R''| = |\ell_j(R) \cap \ell_j(R') \cap \ell_j(R'')|$, *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 $j$th iteration in line 4 of Algorithm 5, $S_1 \cap S_2, S_1 \setminus S_2, S_2 \setminus S_1$ and its images in $\ell_j$ 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 $j$th 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 $\ell_j$ 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 $j$th 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 3.5.1 and definition of $\ell_j$, 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 $\ell_j$ are disjoint so $|R \cap R'| = |\ell_j(R) \cap \ell_j(R)| = 0$. Thus case proven.

✠ *Invariant* IV

Due to the condition in line 9, this invariant is ensured at the end of every iteration.

∎

**Lemma 3.3.4** *If the input ICPPL $(\mathcal{F}, \ell)$ to Algorithm 5 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. Secondly, for any iteration $j > 0$ of the* **while** *loop, the* **exit** *statement in line 10 will not execute.*

**Proof** Since $(\mathcal{F}, \ell)$ is feasible, by Lemma 3.3.2 $(\mathcal{F}_j, \ell_j)$ for every iteration $j > 0$ is feasible. Also, every hypergraph isomorphism $\phi : supp(\mathcal{F}) \rightarrow V(T)$ that induces $\ell$ on $\mathcal{F}$ also induces $\ell_j$ 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 1 outputs a filtered path labeling that "preserves" hypergraph isomorphisms of the original path labeling.
Secondly, line 10 will execute iff the exit condition in line 9, i.e. failure of three way intersection preservation, becomes true in any iteration of the **while** loop. Due to Lemma 3.5.3 Invariant IV, the exit condition does not occur if the input is a feasible ICPPL. ∎

As a result of Algorithm 5 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. In Algorithm 6 we identify 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 $\ell_{new}$ 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 $\ell$ 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 $\ell_{new}$ 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 6 is now presented with details.

**Algorithm 2** Leaf labeling from an ICPPL `filter_2` $(\mathcal{F}, \ell, T)$

```
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:    $\mathcal{F}_j \leftarrow \mathcal{F}_{j-1} \setminus \{S_1\}$
5:    for all $S \in \mathcal{F}_{j-1}$ such that $S \neq S_1$ set $\ell_j(S) \leftarrow \ell_{j-1}(S)$
6:    $X \leftarrow S_1 \setminus \bigcup_{S \in \mathcal{F}_{j-1}, S \neq S_1} S$
7:    if $X$ is empty then
8:       exit
9:    end if
10:   $x \leftarrow$ arbitrary element from $X$
11:   $\mathcal{F}_j \leftarrow \mathcal{F}_j \cup \{\{x\}, S_1 \setminus \{x\}\}$
12:   $\ell_j(\{x\}) \leftarrow \{v\}$
13:   $\ell_j(S_1 \setminus \{x\}) \leftarrow \ell_{j-1}(S_1) \setminus \{v\}$
14:   $j \leftarrow j + 1$
15: end while
16: $\mathcal{F}' \leftarrow \mathcal{F}_j$, $\ell' \leftarrow \ell_j$
17: return $(\mathcal{F}', \ell')$
```

Suppose the input ICPPL $(\mathcal{F}, \ell)$ is feasible, yet set $X$ in Algorithm 6 is empty in some iteration of the **while** loop. This will abort our procedure of finding the hypergraph isomorphism. The following lemma shows that this will not happen.

**Lemma 3.3.5** *If the input ICPPL $(\mathcal{F}, \ell)$ to Algorithm 6 is feasible, then for all iterations $j > 0$ of the* **while** *loop, the* **exit** *statement in line 8 does not execute.*

**Proof** Assume $X$ is empty for some iteration $j > 0$. We know that $v$ is an element of $\ell_{j-1}(S_1)$. Since it is uniquely present in $\ell_{j-1}(S_1)$, it is clear that $v \in \ell_{j-1}(S_1) \setminus \bigcup_{(S \in \mathcal{F}_{j-1}) \wedge (S \neq S_1)} \ell_{j-1}(S)$. Note that for any $x \in S_1$ it is contained in at least two sets due to our assumption about cardinality of $X$. Let $S_2 \in \mathcal{F}_{j-1}$ be another set that contains $x$. From the above argument, we know $v \notin \ell_{j-1}(S_2)$. Therefore there cannot exist a hypergraph isomorphism bijection that maps elements in $S_2$ to those in $\ell_{j-1}(S_2)$. This contradicts our assumption that the input is feasible. Thus $X$ cannot be empty if input is ICPPL and feasible. ∎

**Lemma 3.3.6** *In Algorithm 6, for all $j > 0$, at the end of the $j$th iteration of the* **while** *loop the four invariants given in Lemma 3.5.3 hold.*

**Proof** By Lemma 3.3.5 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 $\ell_j$ 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 as seen in Lemma 3.3.5, $x$ 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. On the other hand, if the input ICPPL is not feasible the invariants are vacuously true. ∎

We have seen two filtering algorithms above, namely, Algorithm 5 `filter_1` and Algorithm 6 `filter_2` which when executed serially repectively result in a new ICPPL on the same universe $U$ and tree $T$. We also proved that if the input is indeed feasible, these algorithms do indeed output the filtered ICPPL. Now we present the algorithmic characterization of a feasible tree path labeling by way of Algorithm 3.

Algorithm 3 computes a hypergraph isomorphism $\phi$ recursively using Algorithm 5 and Algorithm 6 and pruning the leaves of the input tree. In brief, it is done as follows. Algorithm 6 gives us the leaf labels in $\mathcal{F}_2$, i.e., the elements in $supp(\mathcal{F})$ that map to leaves in $T$, where $(\mathcal{F}_2, \ell_2)$ is the output of Algorithm 6. All leaves in $T$ are then pruned away. The leaf labels are removed from the path labeling $\ell_2$ and the corresponding elements are removed from the corresponding sets in $\mathcal{F}_2$. This tree pruning algorithm is recursively called on the altered hypergraph $\mathcal{F}'$, path label $\ell'$ and tree $T'$. The recursive call returns the bijection $\phi''$ for the rest of the elements in $supp(\mathcal{F})$ which along with the leaf labels $\phi'$ gives us the hypergraph isomorphism $\phi$. The following lemma formalizes the characeterization of feasible path labeling.

**Lemma 3.3.7** *If $(\mathcal{F}, \ell)$ is an ICPPL from a tree T and Algorithm 3,* `get- hypergraph-isomorphism` $(\mathcal{F}, \ell, T)$ *returns a non-empty function, then there exists a hypergraph isomorphism $\phi : supp(\mathcal{F}) \to V(T)$ such that the $\phi$-induced tree path labeling is equal to $\ell$ or $\ell_\phi = \ell$.*

**Proof** It is clear that in the end of every recursive call to Algorithm 3, the function $\phi'$ is one to one involving all the leaves in the tree passed as input to that recursive call.

**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_1($\mathcal{F}, \ell, T$)
6: $(\mathcal{F}_2, \ell_2) \leftarrow$ filter_2($\mathcal{F}_1, \ell_1, T$)
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 $\phi' : supp(\mathcal{F}) \rightarrow L$
11:     Remove $\{x\}$ and $\{v\}$ from $\mathcal{F}'$, $\ell'$ 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 $\phi$
```

Moreover, by Lemma 3.3.4 and Lemma 3.3.5 it is consistent with the tree path labeling $\ell$ passed. 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 $\phi : supp(\mathcal{F}) \rightarrow V(T)$ 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 $\ell$ and thus a hypergraph isomorphism. ∎

**Theorem 3.3.8** *A path labeling $(\mathcal{F}, \ell)$ on tree $T$ is feasible iff it is an ICPPL and Algorithm 3 with $(\mathcal{F}, \ell, T)$ as input returns a non-empty function.*

**Proof** From Lemma 3.3.7, we know that if $(\mathcal{F}, \ell)$ is an ICPPL and Algorithm 3 with $(\mathcal{F}, \ell, T)$ as input returns a non-empty function, $(\mathcal{F}, \ell)$ is feasible. Now consider the case where $(\mathcal{F}, \ell)$ is feasible. i.e. there exists a hypergraph isomorphism $\phi$ such that $\ell_\phi = \ell$. Lemma 3.3.4 and Lemma 3.3.5 show us that filter 1 and filter 2 do not exit if input is feasible. Thus Algorithm 3 returns a non-empty function. ∎

## 3.4 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, is a shown to be equivalent to the well studied problem of consecutive-ones in Section 3.4.1. The second one, where the tree is a $k$-subdivided tree[2], has been solved using a polynomial time algorithm. The latter problem enforces some conditions on the hypergraph too which will be seen in Section 3.4.2.

### 3.4.1 Target tree is a Path

Consider a special case of ICPPL with the following properties.

1. Given tree $T$ is a path. Hence, all path labels are interval labels.
2. Only pairwise intersection cardinality preservation is sufficient. i.e. property (iii) in ICPPL is not enforced.
3. The filter algorithms do not have **exit** statements.

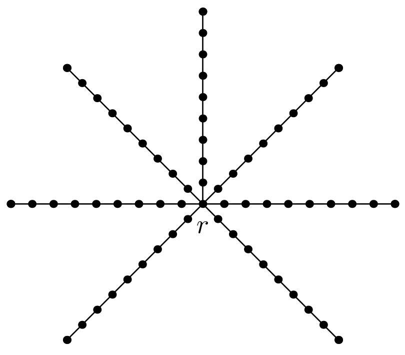
This is called an Intersection Cardinality Preservation Interval Assignment (ICPIA) [NS09]. This structure and its algorithm is used in the next section for finding tree path labeling from a $k$-subdivided star due to this graph's close relationship with intervals.

### 3.4.2 Target tree is a $k$-subdivided Star

In this section we consider the problem of assigning paths from a $k$-subdivided star $T$ to a given set system $\mathcal{F}$. 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

[1]A larger problem would be to find if a given hypergraph is a path graph on any tree. This problem is not addressed in this thesis.

[2]See Section 3.2 for the formal definition.

(a) 

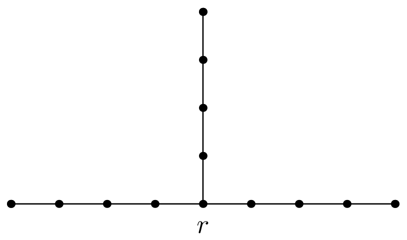
(b) 

Figure 3.1: (a) 8-subdivided star with 7 rays (b) 3-subdivided star with 3 rays

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 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 $R_i$.

In this section the given hypergraph, the $k$-subdivided star and the root of the star are denoted by $\mathcal{O}$, $T$ and vertex $r$, respectively.

The set $\mathcal{O}_i$ refers to the set of hyperedges $\mathcal{T}_1^i \cup \mathcal{T}_2^i$ in the $i$th iteration. Note that $\mathcal{O}_1 = \mathcal{O}$. In the $i$th iteration, hyperedges from $\mathcal{O}_i$ are assigned paths from $R_i$ 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.

Suppose the given hypergraph has a feasible path labeling to the given tree $T$. Let this labeling be $\ell$. Now we present an algorithm that lists the sequence of edges that maintain a set of properties necessary for them to have a feasible tree path labeling.

I. **Step 1**
   (a) **There are type 1 edges, $|\mathcal{T}_1^{\mathbf{i}}| > 0$:** If there is only one $\mathcal{T}_1^i$ hyperedge, label it to the unique path in $R_i$ of length $s(X)$ starting at $v_{i(k+2)} = r$. If there are more than one $\mathcal{T}_1^i$ hyperedges, for any pair of hyperedges $S_1, S_2 \in \mathcal{T}_1^i$, one of the following is true. Let $\ell'(X) \subseteq R_j$ denote the subpath of length $p_1(X)$ that has been assigned to $X$ from a previously considered ray $R_j$.
   *Case* 1. $|\ell'(S_1) \cap \ell'(S_2)| = 1$ and $|S_1 \cap S_2| > 1$ i.e. they are in different rays and their residues must contain their intersection. Hence assign both to $R_i$.
   *Case* 2. $|\ell'(S_1) \cap \ell'(S_2)| > 1$ and $|S_1 \cap S_2| > |\ell'(S_1) \cap \ell'(S_2)|$ i.e. they are in the same rays and their residues must contain their intersection. Hence assign both to $R_i$.
   *Case* 3. $|\ell'(S_1) \cap \ell'(S_2)| > 1$ and $|S_1 \cap S_2| = |\ell'(S_1) \cap \ell'(S_2)|$. i.e. the intersection cardinality has been met. Hence assign only $S_1$ to $R_i$ and $S_2$ must be assigned to some $i \geq p$.

   Remove the labeled edge or edges from $\mathcal{O}_i$ and add it to $\mathcal{O}_{i+1}$. Also do not add it to $\mathcal{T}_1^{i+1}$ and add to $\mathcal{L}_2^{i+1}$.
   (b) **There are no type-1 edges nor super-marginal type-2 hyperedges:** If all type-2 edges of length at most $k+1$, EXIT by saying there is no ICPPL. Otherwise, Find an $X$ of type-2 such that $|X| \geq k+2$, assign it the unique path starting at $v_{i(k+1)} = r$ of length $k+2$, and set $p_1(X) = k+2, s(X) = |X| - (k+2)$, assign it the unique path starting at $v_{i(k+1)} = r$ of length $k+2$, mark this as a type-1 hyperedge, and add it to $\mathcal{O}_{i+1}$ after removing it from $\mathcal{O}_i$.
   (c) **There are super-marginal type-2 hyperedges and no type-1 edges:** Let $X$ be a super-marginal hyperedge. If $|X| \leq k+2$, then $X$ is assigned the path of length $|X|$ starting at $v_{i1}$, and $p_1(X) = |X|, p_2(X) = 0$. $X$ is removed from

$\mathcal{O}_i$ and not added to $\mathcal{O}_{i+1}$. If $|X| > k+2$, then $p_1(X) = k+2$, assign it the unique path starting at $v_{i(k+1)} = r$ of length $k+2$ and $s(X) = |X| - (k+2)$. In this case, $X$ is classified as a Type-1 edge, removed from $\mathcal{O}_i$, and added to $\mathcal{O}_{i+1}$.

II. **Step 2:** Iteratively, a hyperedge $X$ is selected from $\mathcal{O}_i$ that has an overlap with one of the hyperedges $Y$ such that $\ell(Y) \subseteq R_i$, and a unique path is assigned to $X$. The path, say $U(X) \subseteq R_i$, that is assigned can be decided unambiguously since the $X \between Y$, and all intersection cardinalities can be preserved only at one of the ends of $\ell(Y)$.

    Let $\ell(X)$ denote the unique path assigned to $X$. If $X$ is a type-2 hyperedge and if the unique path of length $|X|$ does not contain $r$, then $p_1(X) = |X|, p_2(X) = 0$, and $X$ is removed from $\mathcal{O}_i$. In the case when $\ell(X)$ has to contain $r$, then $p_1(X)$ is the length of the path, $p_2(X) = 0$, and $s(X) = |X| - p_1(X)$. Further $X$ is classified as a type-1 hyperedge, added to $\mathcal{O}_{i+1}$ and removed from $\mathcal{O}_i$. In the case when $X$ is a type-1 hyperedge, then we check if $U(X)$, which is of length $s(X)$ contains $r$. If it does, then we assign $\ell(X) \leftarrow \ell(X) \cup U(X)$, remove $X$ from $\mathcal{O}_i$ and do not add it to $\mathcal{O}_{i+1}$. If not, then we **Exit** reporting that an assignment cannot be found. The iteration ends when no hyperedge in $\mathcal{O}_i$ has an overlap with a hyperedge assigned to $R_i$.

The order in which they are assigned is the sequence of hyperedges with the following properties. Note that if such a sequence cannot be computed, $\mathcal{O}$ cannot have a feasible path labeling from $T$.

In the following lemmas we identify a set of necessary conditions for $\mathcal{F}$ to have an ICPPL in the $k$-subdivided star $T$. If during the execution of the above algorithm, one of these necessary conditions is not satisfied, the algorithm exits reporting the non-existence of an ICPPL.

**Lemma 3.4.1** *Let all hyperedges in $\mathcal{O}_i$ be type-1 edges. Then there is a maximal subset $\mathcal{T}_i \subseteq \mathcal{O}_i$ with the following properties:*

1. *$\mathcal{T}_i$ form an inclusion chain.*
2. *For all $X \in \mathcal{T}_i$, $s(X) \leq k+2$, and There is a an $X \in \mathcal{T}_i$ such that $s(X) = k+2$.*

**Lemma 3.4.2** *If there are no super-marginal type-2 edges in $\mathcal{O}_i$, then there exists at least $p - i$ type-2 hyperedges $X \in \mathcal{O}_i$ such that $|X| \geq k+2$.*

**Lemma 3.4.3** *At the end of Step-1 in the $i$-th iteration, if one hyperedge $X$ of type-2 is such that $\ell(X) \subseteq R_i$, then all other hyperedges in $\mathcal{O}_i$ are connected to $X$ in the overlap component.*

**Lemma 3.4.4** *At the end of Step-2, If control has exit at any time, there is no ICPPL. If control has not exit, then $R_i$ is saturated. No hyperedge of $\mathcal{O}_{i+1}$ will get a path from $R_i$ in the future iterations. No type-2 hyperedge of $\mathcal{O}_{i+1}$ will get a path from $R_i$. Basically $R_i$ is done.*

**Lemma 3.4.5** *Finally, we need to prove that the assignment is an ICPPL. Secondly, if there is a permutation then maps sets to paths, then it is indeed an ICPPL, and our algorithm will basically find it. It is a unique assignment upto permutation of the leaves.*

**Lemma 3.4.6** *If $X \in \mathcal{F}$ is super-marginal and $(\mathcal{F}, \ell)$ is a feasible tree path labeling to tree $T$, then $\ell(X)$ will contain a leaf in $T$.*
*On the otherhand, if $\ell(X)$ has a leaf in $T$, $X$ is marginal but may not be super-marginal.*

**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 $X \not\subseteq Y_1$ 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 proven. ∎

Consider the overlap graph $\mathbb{O}(\mathcal{F})$ of the given hypergraph $\mathcal{F}$. Let $S_{sm} \in \mathcal{F}$ be such that it is a super-marginal hyperedge. Algorithm 4 uses $S_{sm}$ along with the overlap graph $\mathbb{O}(\mathcal{F})$ to calculate the feasible tree path labeling to the $k$-subdivded tree $T$.

## 3.5 TPL with no restrictions

`abstract begin` it is known that if the given tree is a path a feasible assignment can

**Algorithm 4** `compute-ksubtree-path-labeling`($X, \mathcal{F}, T$)

```
1: if $X = S_{sm}$ then
2:     – TBD –
3: else
4:     – TBD –
5: end if
```

be found in polynomial time, and we observe that it can actually be done in logspace.
`abstract end`

In Section 3.5.2 we present the necessary preliminaries, in Section 3.5.3 we present our characterization of feasible tree path assignments, and in Section 3.5.4 we present the characterizing subproblems for finding a bijection between $U$ and $V(T)$ such that sets map to tree paths. Finally, in Section 3.5.5 we conclude by showing that Tree Path Assignment is GI-Complete, and also observe that Consecutive Ones Testing is in Logspace.

We refer to this as the Tree Path Assignment problem for an input $(\mathcal{F}, T)$ pair.

In the second part of this paper, we decompose our search for a bijection between $U$ and $V(T)$ into subproblems. Each subproblem is on a set system in which for each set, there is another set in the set system with which the intersection is *strict*- there is a non-empty intersection, but neither is contained in the other. This is in the spirit of results in [Hsu02, NS09] where to test for COP in a given matrix, the COP problem is solved on an equivalent set of prime matrices. Our decomposition localizes the challenge of path graph isomorphism to two problems.

Finally, we show that Tree Path Assignment is isomorphism-complete. We also point out Consecutive Ones Testing is in Logspace from two different results in the literature [KKLV10, McC04]. To the best of our knowledge this observation has not been made earlier.

In this paper, the collection $\mathcal{F} = \{S_i \mid S_i \subseteq U, S_i \neq \ , i \in I\}$ is a *set system* of universe, $U = \{1, \ldots, n\}$. Moreoever, a set system is assumed to "cover" the universe, i.e. $\bigcup_{i \in I} S_i = U$.

The graph $T = (V, E)$ represents a given tree with $|V| = n$. Further, for simplicity, $V$ is defined as $\{1, \ldots, n\}$. All paths referred to in this paper are paths of $T$ unless explicitly specified.

A *path assignment* $\mathcal{A}$ to $\mathcal{F}$ is defined as a set assignment where second universe is the vertex set $V$ of a given tree $T$ and every second subset in the ordered pairs is a path in this tree. Formally, the definition is as follows.

$$\mathcal{A} = \{(S_i, P_i) \mid S_i \in \mathcal{F}, P_i \subseteq V \text{ s.t. } T[P_i] \text{ is a path}, i \in I\}$$

In other words, $P_i$ is the path on the tree $T$ assigned to $S_i$ in $\mathcal{A}$. As mentioned before for set systems, the paths cover the whole tree, i.e. $\bigcup_{i \in I} P_i = V$

Generalizing the definition of *feasibility* in [NS09] to a set assignment, a path assignment $\mathcal{A}$ is defined to be *feasible* if there exists a bijection defined as follows.

$$\sigma : U \to V(T), \text{ such that } \sigma(S_i) = P_i \text{ for all } i \in I, \sigma \text{ is a bijection} \qquad (3.1)$$

Let $X$ be a partially ordered set with $\preccurlyeq$ being the partial order on $X$. $mub(X)$ represents an element in $X$ which is a maximal upper bound on $X$. $X_m \in X$ is a maximal upper bound of $X$ if $\nexists X_q \in X$ such that $X_m \preccurlyeq X_q$.

The set $I$ represents the index set $[m]$. If index $i$ is used without further qualification, it is meant to be $i \in I$. Any function, if not defined on a domain of sets, when applied on a set is understood as the function applied to each of its elements. i.e. for any function $f$ defined with domain $U$, the abuse of notation is as follows; $f(S)$ is used instead of $\hat{f}(S)$ where $\hat{f}(S) = \{y \mid y = f(x), x \in S\}$.

When refering to a tree as $T$ it could be a reference to the tree itself, or the vertices of the tree. This will be clear from the context.

Finally, an in-tree is a directed rooted tree in which all edges are directed toward to the root.

Consider a path assignment $\mathcal{A} = \{(S_i, P_i) \mid S_i \in \mathcal{F}, P_i$ is a path from $T, i \in [m]\}$ to a set system $\mathcal{F} = \{S_i \mid S_i \subseteq U, i \in [m]\}$, were $T$ is a given tree, $U$ is the set system's universe and $m$ is the number of sets in $\mathcal{F}$. We call $\mathcal{A}$ an *Intersection Cardinality Preserving Path Assignment (ICPPA)* if it has the following properties.

i. $|S_i| = |P_i|$ for all $i \in [m]$

ii. $|S_i \cap S_j| = |P_i \cap P_j|$ for all $i, j \in [m]$

iii. $|S_i \cap S_j \cap S_k| = |P_i \cap P_j \cap P_k|$ for all $i, j, k \in [m]$

**Lemma 3.5.1** *If $\mathcal{A}$ is an ICPPA, and $(S_1, P_1), (S_2, P_2), (S_3, P_3) \in \mathcal{A}$, then $|S_1 \cap (S_2 \setminus S_3)| = |P_1 \cap (P_2 \setminus P_3)|$.*

**Proof** $|S_1 \cap (S_2 \setminus S_3)| = |(S_1 \cap S_2) \setminus S_3| = |S_1 \cap S_2| - |S_1 \cap S_2 \cap S_3|$. Due to conditions (ii) and (iii) of ICPPA, $|S_1 \cap S_2| - |S_1 \cap S_2 \cap S_3| = |P_1 \cap P_2| - |P_1 \cap P_2 \cap P_3| = |(P_1 \cap P_2) \setminus P_3| = |P_1 \cap (P_2 \setminus P_3)|$. Thus lemma is proven. ∎

**Lemma 3.5.2** *Consider four paths in a tree $Q_1, Q_2, Q_3, Q_4$ such that they have nonempty pairwise intersection and $Q_1, Q_2$ share a leaf. Then there exists distinct $i, j, k \in \{1, 2, 3, 4\}$ such that, $Q_1 \cap Q_2 \cap Q_3 \cap Q_4 = Q_i \cap Q_j \cap Q_k$.*

**Proof** *Case 1:* w.l.o.g, consider $Q_3 \cap Q_4$ and let us call it $Q$. This is clearly a path (intersection of two paths is a path). Suppose $Q$ does not intersect with $Q_1 \setminus Q_2$, i.e. $Q \cap (Q_1 \setminus Q_2) =$ . Then $Q \cap Q_1 \cap Q_2 = Q \cap Q_2$. Similarly, if $Q \cap (Q_2 \setminus Q_1) =$ , $Q \cap Q_1 \cap Q_2 = Q \cap Q_1$. Thus it is clear that if the intersection of any two paths does not intersect with any of the set differences of the remaining two paths, the claim in the lemma is true. *Case 2:* Let us consider the compliment of the previous case. i.e. the intersection of any two paths intersects with both the set differences of the other two. First let us consider $Q \cap (Q_1 \setminus Q_2) \neq$ and $Q \cap (Q_1 \setminus Q_2) \neq$ , where $Q = Q_3 \cap Q_4$. Since $Q_1$ and $Q_2$ share a leaf, there is exactly one vertex at which they branch off from the path $Q_1 \cap Q_2$ into two paths $Q_1 \setminus Q_2$ and $Q_2 \setminus Q_1$. Let this vertex be $v$. It is clear that if path $Q_3 \cap Q_4$, must intersect with paths $Q_1 \setminus Q_2$ and $Q_2 \setminus Q_1$, it must contain $v$

since these are paths from a tree. Moreover, $Q_3 \cap Q_4$ intersects with $Q_1 \cap Q_2$ at exactly $v$ and only at $v$ which means that $Q_1 \cap Q_2$ does not intersect with $Q_3 \setminus Q_4$ or $Q_4 \setminus Q_3$ which contradicts initial condition of this case. Thus this case cannot occur and case 1 is the only possible scenario.
Thus lemma is proven ∎

In the remaining part of this section we show that a path assignment is feasible if and only if it is an ICPPA. One direction of this claim is clear: that is a path assignment is feasible, then all intersection cardinalities are preserved, that is the path assignment is an ICPPA. The reason is that a feasible path assignment has an associated bijection between $U$ and $V(T)$ such that the sets map to paths. The rest of the section is devoted to constructively proving that it is sufficient, for the existence of a bijection, for a path assignment to be an ICPPA. At a top-level, the constructive approaches refine the path assignment iteratively, such that at the end of each iteration we have a path assignment, and finally we have a family of bijections. First we present and then prove the correctness of Algorithm 5. This algorithm refines the path assignment by considering pairs of paths that share a leaf.

**Algorithm 5** Permutations from an ICPPA $\{(S_i, P_i) | i \in I\}$

```
Let $\Pi_0 = \{(S_i, P_i) | i \in I\}$
$j = 1$;
**while** There is $(P_1, Q_1), (P_2, Q_2) \in \Pi_{j-1}$ with $Q_1$ and $Q_2$ having a common leaf **do**
    $\Pi_j = \Pi_{j-1} \setminus \{(P_1, Q_1), (P_2, Q_2)\}$;
    $\Pi_j = \Pi_j \cup \{(P_1 \cap P_2, Q_1 \cap Q_2), (P_1 \setminus P_2, Q_1 \setminus Q_2), (P_2 \setminus P_1, Q_2 \setminus Q_1)\}$;
    $j = j + 1$;
**end while**
$\Pi = \Pi_j$;
Return $\Pi$;
```

**Lemma 3.5.3** *In Algorithm 5, at the end of jth iteration, $j \geq 0$, of the while loop of Algorithm 5, the following invariants are maintained.*

- Invariant I: *$Q$ is a path in $T$ for each $(P, Q) \in \Pi_j$*
- Invariant II: $|P| = |Q|$ *for each* $(P, Q) \in \Pi_j$
- Invariant III: *For any two* $(P, Q), (P', Q') \in \Pi_j$, $|P' \cap P''| = |Q' \cap Q''|$.
- Invariant IV: *For any three,* $(P', Q'), (P'', Q''), (P, Q) \in \Pi_j$, $|P' \cap P'' \cap P| = |Q' \cap Q'' \cap Q|$.

**Proof** Proof is by induction on the number of iterations, $j$. In the rest of the proof, the term "new sets" will refer to the new sets added in $j$th iteration as defined in line 4 of Algorithm 5, i.e. the following three assignment pairs for some $(P_1, Q_1), (P_2, Q_2) \in \Pi_{j-1}$ where $Q_1$ and $Q_2$ intersect and share a leaf: $(P_1 \cap P_2, Q_1 \cap Q_2)$, or $(P_1 \setminus P_2, Q_1 \setminus Q_2)$, or $(P_2 \setminus P_1, Q_2 \setminus Q_1)$.
The base case, $\Pi_0 = \{(S_i, P_i) \mid i \in [m]\}$, is trivially true since it is the input which is an ICPPA. Assume the lemma is true till the $j-1$ iteration. Consider $j$th iteration:
If $(P, Q)$, $(P', Q')$ and $(P'', Q'')$ are in $\Pi_j$ and $\Pi_{j-1}$, all the invariants are clearly true because they are from $j-1$ iteration.
If $(P, Q)$ is in $\Pi_j$ and not in $\Pi_{j-1}$, then it must be one of the new sets added in $\Pi_j$. Since $(P_1, Q_1)$ and $(P_2, Q_2)$ are from $\Pi_{j-1}$ and $Q_1, Q_2$ intersect and have a common leaf, it can be verified that the new sets are also paths.
By hypothesis for invariant III, invariant II also holds for $(P, Q)$ no matter which new set in $\Pi_j$ it is.
To prove invariant III, if $(P, Q)$ and $(P', Q')$ are not in $\Pi_{j-1}$, then they are both new sets and invariant III holds trivially (new sets are disjoint). Next consider $(P, Q), (P', Q') \in \Pi_j$ with only one of them, say $(P', Q')$, in $\Pi_{j-1}$. Then $(P, Q)$ is one of the new sets added in line 4. It is easy to see that if $(P, Q)$ is $(P_1 \cap P_2, Q_1 \cap Q_2)$, then due to invariant IV in hypothesis, invariant III becomes true in this iteration. Similarly, using lemma 3.5.1 invariant III is proven if $(P, Q)$ is $(P_1 \setminus P_2, Q_1 \setminus Q_2)$, or $(P_2 \setminus P_1, Q_2 \setminus Q_1)$.
To prove invariant IV, consider three assignments $(P, Q), (P', Q'), (P'', Q'')$. If at least two of these pairs are in not $\Pi_{j-1}$, then they are any two of the new sets. Note that these new sets are disjoint and hence if $(P', Q'), (P'', Q'')$ are any of these sets, $|P \cap P' \cap P''| = |Q \cap Q' \cap Q''| = 0$ and invariant IV is true. Now we consider the case if at most one of $(P, Q), (P', Q'), (P'', Q'')$ is not in $\Pi_{j-1}$. If none of them are not in $\Pi_{j-1}$ (i.e. all of them are in $\Pi_{j-1}$), invariant IV is clearly true. Consider the case where exactly one of them is not in $\Pi_{j-1}$. w.l.o.g let that be $(P, Q)$ and it could be any one of the new sets. If $(P, Q)$ is $(P_1 \cap P_2, Q_1 \cap Q_2)$, from lemma 3.5.2 and invariant III hypothesis, invariant IV is proven. Similarly if $(P, Q)$ is any of the other new sets, invariant IV is proven by also using lemma 3.5.1. ∎

It can be observed that the output of algorithm 5 is such that every leaf is incident on at most a single path in the new set of assignments. This is due to the loop condition at line 2. Let $v_1$ be the leaf incident on path $P_i$. Assign to it any one element from $S_i \setminus \bigcup_{i \neq j} S_j$. Remove $(S_i, P_i)$ from assignments and add $(\{x_1\}, \{v_1\}), (S_i \setminus \{x_1\}, P_i \setminus \{v_1\})$. Now all assignments except single leaf assignments are paths from the subtree $T_0 = T \setminus \{v \mid v \text{ is a leaf in } T\}$.

**Algorithm 6** Leaf assignments from an ICPPA $\{(S_i, P_i) | i \in I\}$

```
Let $\Pi_0 = \{(S_i, P_i) | i \in [m]\}$. Paths are such that no two paths $P_i, P_j, i \neq j$ share a leaf.
$j = 1$
while there is a leaf $v$ and a unique $(S_{i_1}, P_{i_1})$ such that $v \in P_{i_1}$ do
    $\Pi_j = \Pi_{j-1} \setminus \{(S_{i_1}, P_{i_1})\}$
    $X = S_{i_1} \setminus \bigcup_{i \neq i_1, i \in I} S_i$
    if $X$ is empty then
        exit
    end if;
    Let $x =$ arbitrary element from $X$
    $\Pi_j = \Pi_j \cup \{(S_{i_1} \setminus \{x\}), (P_{i_1} \setminus \{v\}), (\{x\}, \{v\})\}$
    $j = j + 1$
end while
$\Pi = \Pi_j$
Return $\Pi$
```

**Lemma 3.5.4** *In Algorithm 6,for all $j \geq 0$, at the end of the $j$th iteration the four invariants given in lemma 3.5.3 are valid.*

**Proof** First we see that $X = S_{i_1} \setminus \bigcup_{i \neq i_1, i \in I} S_i$ is non empty in every iteration for an ICPPA. Suppose $X$ is empty. We know that $v \in P_{i_1} \setminus \bigcup_{i \neq i_1, i \in I} P_i$ since $v$ is in the unique path $P_{i_1}$. Since this is an ICPPA $|S_{i_1}| = |P_{i_1}|$. For any $x \in S_{i_1}$ it is contained in at least two sets due to our assumption. Let $S_{i_2}$ be a second set that contains $x$. We know $v \notin P_{i_2}$. Therefore there cannot exist a permutation that maps elements of $S_{i_2}$ to $P_{i_2}$. This contradicts our assumption that this is an ICPPA. Thus $X$ cannot be empty.

We use mathematical induction on the number of iterations for this proof. The term "new sets" will refer to the sets added in $\Pi_j$ in the $j$th iteration, i.e. $(P' \setminus \{x\}, Q' \setminus \{v\})$ and $(\{x\}, \{v\})$ for some $(P', Q')$ in $\Pi_{j-1}$ such that $v$ is a leaf and $Q'$ is the unique path incident on it.

For $\Pi_0$ all invariants hold because it is output from algorithm 5 which is an ICPPA.

Hence base case is proved. Assume the lemma holds for $\Pi_{j-1}$. Consider $\Pi_j$ and any $(P, Q) \in \Pi_j$. If $(P, Q)$ is in $\Pi_j$ and $\Pi_{j-1}$ invariants I and II are true because of induction assumption. If it is only in $\Pi_j$, then it is $\{(P' \setminus \{x\}), (Q' \setminus \{v\})$ or $(\{x\}, \{v\})$ for some $(P', Q')$ in $\Pi_{j-1}$. By definition, $x$ is an element in $P'$ (as defined in the algorithm) and $v$ is a leaf in $Q'$. If $(P, Q)$ is $\{(P' \setminus \{x\}), (Q' \setminus \{v\})$, $Q$ is a path since only a leaf is removed from path $Q'$. We know $|P'| = |Q'|$, therefore $|P' \setminus \{x\}| = |Q' \setminus \{v\}|$. Hence in this case invariants I and II are obvious. It is easy to see these invariants hold if $(P, Q)$ is $(\{x\}, \{v\})$.

For invariant III consider $(P_1, Q_1), (P_2, Q_2)$ in $\Pi_j$. If both of them are also in $\Pi_{j-1}$, claim is proved. If one of them is not in $\Pi_{j-1}$ then it has to be $\{(P' \setminus \{x\}), (Q' \setminus \{v\})$ or $(\{x\}, \{v\})$ for some $(P', Q')$ in $\Pi_{j-1}$. Since by definition, $Q'$ is the only path with $v$ and $P'$ the only set with $x$ in the previous iteration, $|P_1 \cap (P' \setminus \{x\})| = |P_1 \cap P'|$ and $|Q_1 \cap (Q' \setminus \{v\})| = |Q_1 \cap Q'|$ and $|P_1 \cap \{x\}| = 0, Q_1 \cap \{v\} = 0$. Thus invariant III is also proven.

To prove invariant IV, consider $(P_1, Q_1), (P_2, Q_2), (P_3, Q_3)$ in $\Pi_j$. If exactly one of them, say $P_3 \notin \Pi_{j-1}$, it is one of the new sets. By the same argument used to prove invariant III, $|P_1 \cap P_2 \cap (P' \setminus \{x\})| = |P_1 \cap P_2 \cap P'|$ and $|Q_1 \cap Q_2 \cap (Q' \setminus \{x\})| = |Q_1 \cap Q_2 \cap Q'|$. Since $P_1, P_2, P'$ are all in $\Pi_{j-1}$, by induction hypothesis $|P_1 \cap P_2 \cap P'| = |Q_1 \cap Q_2 \cap Q'|$. Also $|P_1 \cap P_2 \cap \{x\}| = 0, Q_1 \cap Q_2 \cap \{v\} = 0$. If two or more of them are not in $\Pi_{j-1}$, then it can be verified that $|P_1 \cap P_2 \cap P_3| = |Q_1 \cap Q_2 \cap Q_3|$ since the new sets in $\Pi_j$ are either disjoint or as follows: assuming $P_1, P_2 \notin \Pi_{j-1}$ and new sets are derived from $(P', Q'), (P'', Q'') \in \Pi_{j-1}$ with $x_1, x_2$ exclusively in $P_1, P_2$, $(\{x_1\}, \{v_1\}), (\{x_2\}, \{v_2\}) \in \Pi_j$ thus $v_1, v_2$ are exclusively in $Q_1, Q_2$ resp. it follows that $|P_1 \cap P_2 \cap P_3| = |(P' \setminus \{x_1\}) \cap (P'' \setminus \{x_2\}) \cap P_3| = |P' \cap P'' \cap P_3| = |Q' \cap Q'' \cap Q_3| = |(Q' \setminus \{v_1\} \cap Q'' \setminus \{v_2\} \cap Q_3| = |Q_1 \cap Q_2 \cap Q_3|$. Thus invariant IV is also proven. ∎

Using algorithms 5 and 6 we prove the following theorem.

**Theorem 3.5.5** *If $\mathcal{A}$ is an ICPPA, then there exists a bijection $\sigma : U \to V(T)$ such that $\sigma(S_i) = P_i$ for all $i \in I$*

**Proof** This is a contructive proof. First, the given ICPPA $\mathcal{A}$ and tree $T$ are given as input to Algorithm 5. This yields a "filtered" ICPPA as the output which is input to Algorithm 6. It can be observed that the output of Algorithm 6 is a set of interval assignments to sets and one-to-one assignment of elements of $U$ to each leaf of $T$. To be precise, it would be of the form $\mathcal{B}_0 = \mathcal{A}_0 \cup \mathcal{L}_0$. The leaf assignments are defined in $\mathcal{L}_0 = \{(x_i, v_i) \mid x_i \in U, v_i \in T, x_i \neq x_j, v_i \neq v_j, i \neq j, i, j \in [k]\}$ where $k$ is the number of leaves in $T$. The path assignments are defined in $\mathcal{A}_0 \subseteq \{(S'_i, P'_i) \mid S'_i \subseteq U_0, P'_i$ is a path from $T_0\}$ where $T_0$ is the tree obtained by removing all the leaves in $T$ and $U_0 = U \setminus \{x \mid x$ is assigned to a leaf in $\mathcal{L}_0\}$. Now we have a subproblem of finding the permutation for the path assignment $\mathcal{A}_0$ which has paths from tree $T_0$ and sets from universe $U_0$. Now we repeat the procedure and the path assignment $\mathcal{A}_0$ and tree $T_0$ is given as input to Algorithm 5. The output of this algorithm is given to Algorithm 6 to get a new union of path and leaf assignments $\mathcal{B}_1 = \mathcal{A}_1 \cup \mathcal{L}_1$ defined similar to $\mathcal{B}_0, \mathcal{L}_0, \mathcal{A}_0$. In general, the two algorithms are run on path assignment $\mathcal{A}_{i-1}$ with paths from tree $T_{i-1}$ to get a new subproblem with path assignment $\mathcal{A}_i$ and tree $T_i$. $T_i$ is the subtree of $T_{i-1}$ obtained by removing all its leaves. More importantly, it gives leaf assignments $\mathcal{L}_i$ to the leaves in tree $T_{i-1}$. This is continued until we get a subproblem with path assignment $\mathcal{A}_{d-1}$ and tree $T_{d-1}$ for some $d \leq n$ which is just a path. From the last lemma we know that $\mathcal{A}_{d-1}$ is an ICPPA. Another observation is that an ICPPA with all its tree paths being intervals (subpaths from a path) is nothing but an ICPIA[NS09]. Let $\mathcal{A}_{d-1}$ be equal to $\{(S''_i, P''_i) \mid S''_i \subseteq U_{d-1}, P''_i$ is a path from $T_{d-1}\}$. It is true that the paths $P''_i$s may not be precisely an interval in the sense of consecutive integers because they are some nodes from a tree. However, it is easy to see that the nodes of $T_{d-1}$ can be ordered from left to right and ranked to get intervals $I_i$ for every path $P''_i$ as follows. $I_i = \{[l, r] \mid l =$ the lowest rank of the nodes in $P''_i, r = l + |P''_i| - 1\}$. Let asssignment $\mathcal{A}_d$ be with the renamed paths. $\mathcal{A}_d = \{(S''_i, I_i) \mid (S''_i, P''_i) \in \mathcal{A}_{d-1}\}$. What has been effectively done is renaming the nodes in $T_{d-1}$ to get a tree $T_d$. The ICPIA $\mathcal{A}_d$ is now in the format that the ICPIA algorithm requires which gives us the permutation $\sigma' : U_{d-1} \rightarrow T_{d-1}$

$\sigma'$ along with all the leaf assignments $\mathcal{L}_i$ gives us the permutation for the original path assignment $\mathcal{A}$. More precisely, the permutation for tree path assignment $\mathcal{A}$ is defined as follows. $\sigma : U \rightarrow T$ such that the following is maintained.

$$\begin{aligned}\sigma(x) &= \sigma'(x), \text{ if } x \in U_{d-1} \\ &= \mathcal{L}_i(x), \text{ where } x \text{ is assigned to a leaf in a subproblem } \mathcal{A}_{i-1}, T_{i-1}\end{aligned}$$

To summarize, run algorithm 5 and 6 on $T$. After the leaves have been assigned to specific elements from $U$, remove all leaves from $T$ to get new tree $T_0$. The leaf assignments are in $\mathcal{L}_0$. Since only leaves were removed $T_0$ is indeed a tree. Repeat the algorithms on $T_0$ to get leaf assignments $\mathcal{L}_1$. Remove the leaves in $T_0$ to get $T_1$ and so on until the pruned tree $T_d$ is a single path. Now run ICPIA algorithm on $T_d$ to get permutation $\sigma'$. The relation $\mathcal{L}_0 \cup \mathcal{L}_1 \cup .. \cup \mathcal{L}_d \cup \sigma'$ gives the bijection required in the original problem. ∎

### 3.5.1 Finding an assignment of tree paths to a set system

In the previous section we have shown that the problem of finding a Tree Path Asssignment to an input $(\mathcal{F}, T)$ is equivalent to finding an ICPPA to $\mathcal{F}$ in tree $T$. In this section we characterize those set systems that have an ICPPA in a given tree. As a consequence of this characterization we identify two sub-problems that must be solved to obtain an ICPPA. We do not solve the problem and in the next section show that finding an ICPPA in a given tree is GI-Complete.

A set system can be concisely represented by a binary matrix where the row indices denote the universe of the set system and the column indices denote each of the sets. Let the binary matrix be $M$ with order $n \times m$, the set system be $\mathcal{F} = \{S_i \mid i \in [m]\}$, universe of set system $U = \{x_1, \ldots, x_n\}$. If $M$ represents $\mathcal{F}$, $|U| = n, |\mathcal{F}| = m$. Thus $(i, j)$th element of $M$, $M_{ij} = 1$ iff $x_i \in S_j$. If $\mathcal{F}$ has a feasible tree path assignment (ICPPA) $\mathcal{A} = \{(S_i, P_i) \mid i \in [m]\}$, then we say its corresponding matrix $M$ has an ICPPA. Conversly we say that a matrix $M$ has an ICPPA if there exists an ICPPA $\mathcal{A}$ as defined above.

We now define the strict intersection graph or overlap graph of $\mathcal{F}$. This graph occurs at many places in the literature, see for example [KKLV10, Hsu02, NS09]. The vertices of the graph correspond to the sets in $\mathcal{F}$. An edge is present between vertices of two sets iff the corresponding sets have a nonempty intersection and none is contained in the other. Formally, intersection graph is $G_f = (V_f, E_f)$ such that $V_f = \{v_i \mid S_i \in \mathcal{F}\}$ and $E_f = \{(v_i, v_j) \mid S_i \cap S_j \neq \text{ and } S_i \nsubseteq S_j, S_j \nsubseteq S_i\}$. We use this graph to decompose $M$ as described in [Hsu02, NS09]. A prime sub-matrix of $M$ is defined as the matrix formed by a set of columns of $M$ which correspond to a connected component of the graph $G_f$. Let us denote the prime sub-matrices by $M_1, \ldots, M_p$ each corresponding to one of the $p$ components of $G_f$. Clearly, two distinct matrices have a distinct set of columns. Let $col(M_i)$ be the set of columns in the sub-matrix $M_i$. The support of a prime sub-matrix $M_i$ is defined as $supp(M_i) = \bigcup\limits_{j \in col(M_i)} S_j$. Note that for each $i$, $supp(M_i) \subseteq U$. For a set of prime sub-matrices $X$ we define $supp(X) = \bigcup\limits_{M \in X} supp(M)$.

Consider the relation $\preccurlyeq$ on the prime sub-matrices $M_1, \ldots, M_p$ defined as follows:

$$\{(M_i, M_j) \mid \text{A set } S \in M_i \text{ is contained in a set } S' \in M_j\} \cup \{(M_i, M_i) \mid 1 \leq i \leq p\}$$

This relation is the same as that defined in [NS09]. The prime submatrices and the above relation can be defined for any set system. We will use this structure of prime submatrices to present our results on an an ICPPA for a set system $\mathcal{F}$. Recall the following lemmas, and theorem that $\preccurlyeq$ is a partial order, from [NS09].

**Lemma 3.5.6** *Let $(M_i, M_j) \in \preccurlyeq$. Then there is a set $S' \in M_j$ such that for each $S \in M_i$, $S \subseteq S'$.*

**Lemma 3.5.7** *For each pair of prime sub-matrices, either $(M_i, M_j) \notin \preccurlyeq$ or $(M_j, M_i) \notin \preccurlyeq$.*

**Lemma 3.5.8** *If $(M_i, M_j) \in \preccurlyeq$ and $(M_j, M_k) \in \preccurlyeq$, then $(M_i, M_k) \in \preccurlyeq$.*

**Lemma 3.5.9** *If $(M_i, M_j) \in \preccurlyeq$ and $(M_i, M_k) \in \preccurlyeq$, then either $(M_j, M_k) \in \preccurlyeq$ or $(M_k, M_j) \in \preccurlyeq$.*

**Theorem 3.5.10** *$\preccurlyeq$ is a partial order on the set of prime sub-matrices of $M$. Further, it uniquely partitions the prime sub-matrices of $M$ such that on each set in the partition $\preccurlyeq$ induces a total order.*

For the purposes of this paper, we refine the total order mentioned in Theorem 3.5.10. We do this by identifying an in-tree rooted at each maximal upper bound under $\preccurlyeq$. Each of these in-trees will be on disjoint vertex sets, which in this case would be disjoint sets of prime-submatrices. The in-trees are specified by selecting the appropriate edges from the Hasse diagram associated with $\preccurlyeq$. Let $\mathcal{I}$ be the following set:

$$\mathcal{I} = \{(M_i, M_j) \in \preccurlyeq \mid \nexists M_k s.t. M_i \preccurlyeq M_k, M_k \preccurlyeq M_j\} \cup \{(M_i, M_i), i \in [p]\}$$

**Theorem 3.5.11** *Consider the directed graph $X$ whose vertices correspond to the prime sub-matrices, and the edges are given by $\mathcal{I}$. Then, $X$ 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 out-going edge is present in $I$. Secondly, for each other element, exactly one out-going edge is chosen, and for the minimal lower bound, there is no in-coming edge. Consequently, $X$ is acyclic, and since each vertex has at most one edge leaving it, it follows that $X$ is a collection of in-trees, and for each in-tree, the root is a maximal upper bound in $\preccurlyeq$. Hence the theorem.

Let the partition of $X$ given by Theorem 3.5.11 be $\{X_1, \ldots, X_r\}$. Further, each in-tree itself can be layered based on the distance from the root. The root is considered to be at level zero. For $j \geq 0$, Let $X_{i,j}$ denote the set of prime matrices in level $j$ of in-tree $X_i$.

**Lemma 3.5.12** *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 $\{X_1, \ldots, X_r\}$ be the partition of $X$ into in-trees as defined above. Then, matrix $M$ has an ICPPA in tree $T$ iff $T$ can be partitioned into vertex disjoint subtrees $\{T_1, T_2, \ldots T_r\}$ such that, for each $1 \leq i \leq r$, the set of prime sub-matrices corresponding to vertices in $X_i$ has an ICPPA in $T_i$.*

**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 $1 \leq i \leq r$, the set of prime sub-matrices corresponding to vertices in $X_i$ has an ICPPA in $T_i$. It is clear from the properties of the partial order $\preccurlyeq$ that these ICPPAs naturally yield an ICPPA of $M$ in $T$. The main property used in this inference is that for each $1 \leq i \neq j \leq r$, $supp(X_i) \cap supp(X_j) = \phi$.

To prove the forward direction, we show that if $M$ has an ICPPA, say $\mathcal{A}$, in $T$, then there exists a partition of $T$ into vertex disjoint subtree $T_1, \ldots, T_r$ such that for each $1 \leq i \leq r$, the set of prime sub-matrices corresponding to vertices in $X_i$ has an ICPPA in $T_i$. For each $1 \leq i \leq r$, we define based on $\mathcal{A}$ a subtree $T_i$ corresponding to $X_i$. We then argue that the trees thus defined are vertex disjoint, and complete the proof. Consider $X_i$ and consider the prime sub-matrix in $X_{i,0}$. Consider the paths assigned under $\mathcal{A}$ to the sets in the prime sub-matrix in $X_{i,0}$. Since the component in $G_f$ corresponding to this matrix is a connected component, it follows that union of paths assigned to this prime-submatrix is a subtree of $T$. We call this sub-tree $T_i$. All other prime-submatrices in $X_i$ are assigned paths in $T_i$ since $\mathcal{A}$ is an ICPPA, and the support of other prime sub-matrices in $X_i$ are contained in the support of the matrix in $X_{i,0}$. Secondly, for each $1 \leq i \neq j \leq r$, $supp(X_i) \cap supp(X_j) = \phi$, and since $\mathcal{A}$ is an ICPPA, 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 a partition of $T$ into vertex disjoint sub-trees such that for each $1 \leq i \leq r$, the set of matrices corresponding to nodes in $X_i$ has an ICPPA in $T_i$. Hence the lemma.

The essence of the following lemma is that an ICPPA only needs to be assigned to the prime sub-matrix 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 pseudo-code to test if $M$ has an ICPPA in $T$.

**Lemma 3.5.13** *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 $\{X_1, \ldots, X_r\}$ be the partition of $X$ into in-trees as defined earlier in this section. Let $T$ be the given tree and let $\{T_1, \ldots, T_r\}$ be a given partition of $T$ into vertex disjoint sub-trees. Then, for each $1 \leq i \leq r$, the set of matrices corresponding to vertices of $X_i$ has an ICPPA in $T_i$ if and only if the matrix in $X_{i,0}$ has an ICPPA in $T_i$ and all other matrices in $X_i$ have an* ***ICPIA****.*

**Proof** The proof is based on the following fact- $\preccurlyeq$ is a partial order and $X$ is a digraph which is the disjoint union of in-trees. Each edge in the in-tree is a containment relationship among the supports of the corresonding sub-matrices. Therefore, any ICPPA to a prime sub-matrix that is not the root is contained in a path assigned to the sets in the parent matrix. Consequently, any ICPPA to the prime sub-matrix that is not at the root is an ICPIA, and any ICPIA can be used to construct an ICPPA to the matrices corresponding to nodes in $X_i$ provided the matrix in the root has an ICPPA in $T_i$. Hence the lemma.

Lemma 3.5.12 and Lemma 3.5.13 point out two algorithmic challenges in finding an ICPPA for a given set system $\mathcal{F}$ in a tree $T$. Given $\mathcal{F}$, finding $X$ and its partition $\{X_1, \ldots, X_r\}$ into in-trees can be done in polynomial time. On the other hand, as per lemma 3.5.12 we need to parition $T$ into vertex disjoint sub-trees $\{T_1, \ldots, T_r\}$ such that for each $i$, the set of matrices corresponding to nodes in $X_i$ have an ICPPA in $T_i$. 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. The second algorithmic challenge is identified by lemma 3.5.13 which is to assign an ICPPA from a given tree to the matrix associated with the root node of $X_i$.

### 3.5.2 Complexity of Tree Path Assignment-A Discussion

Recall that the input to the Tree Path Assignment question is an order pair $(\mathcal{F}, T)$ where $\mathcal{F}$ is a family of subsets of an universe $U$, and $T$ is a tree such that $|V(T)| = |U|$. The question is to come up with a bijection from $U$ to $V(T)$ such that the image of each

**Algorithm 7** Algorithm to find an ICPPA for a matrix $M$ on tree $T$: $main\_ICPPA(M, T)$

```
Identify the prime sub-matrices. This is done by constructing the strict overlap graph and identify connected components. Each connected component yields a prime sub-matrix.
Construct the partial order $\preccurlyeq$ on the set of prime sub-matrices.
Construct the partition $X_1, \ldots, X_r$ of the prime sub-matrices induced by $\preccurlyeq$
For each $1 \leq i \leq r$, Check if all matrices except those in $X_{i,0}$ 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].
Find a partition of $T_1, \ldots, T_r$ such that matrices in $X_{i,0}$ has an ICPPA in $T_i$. If not such partition exists, exit with negative answer.
```

set in $\mathcal{F}$ is a path in $T$. We show that this problem is at least as hard as the problem of testing if two given path chordal graphs are isomorphic.

**Theorem 3.5.14** *Tree Path Assignment is isomorphism-complete.*

**Proof** It is well known (see for example [KKLV10]) that testing isomorphism of path chordal graphs is isomorphism complete. We show a reduction of path chordal graph isomorphism to tree path assignment. Given $G_1$ and $G_2$ two path chordal graphs, let $T_2$ be the clique tree of $G_2$ obtained from say [Gav78]. The nodes of $T_2$ correspond to the maximal cliques of $G_2$ and each vertex of $G_2$ corresponds to a path in $G_2$. This is a well-known characterization of path chordal graphs and $T_2$ can be computed in polynomial time. In $G_1$, let $S_v$ denote the maximal cliques of $G_1$ that contain $v$. This can be computed in polynomial time as $G_1$ is path chordal, and all chordal graph only have a linear number of maximal cliques. The universe $U$ corresponds bijectively to the set of maximal cliques in $G_1$, and $\mathcal{F} = \{S_v | v \in V(G_1)\}$. Now, we claim that $(\mathcal{F}, T_2)$ has a tree path assignment if and only if $G_1$ and $G_2$ are isomorphic. This is clear since for each vertex $v \in G_1$, there is an associated $S_v$ which is the set of maximal cliques containing $v$. In $G_2$, each vertex corresponds to a path in $T_2$, and the nodes on this path corresponds to the maximal cliques in $G_2$. Consequenlty, a tree path assignment will naturally yield an isomorphism between $G_1$ and $G_2$, and vice versa. Therefore, Tree Path Assignment is isomorphism-complete.

**Consecutive Ones Testing is in Logspace**

While Tree Path Assignment is isomorphism-complete, it is polynomial time solvable when the given tree is a path. Indeed, in this case we encounter a restatement of matrices with the COP. 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. The first linear time algorithm for testing COP for a binary matrix was using a data structure called PQ trees, which represent all COP orderings of $M$, invented by [BL76]. There is a PQ tree for a matrix iff the matrix has COP. Indeed, this is an algorithmic characterization of the consecutive ones property and the absence of the PQ-tree does not yield any witness to the reason for failure. A closely related data structure is the generalized PQ tree in [McC04]. In generalized PQ tree the 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 [McC04]. 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. [McC04] describes a graph called incompatibility graph of a set system $\mathcal{F}$ which 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 $\mathcal{F}$.

**Theorem 3.5.15 (Theorem 6.1, [McC04])** *Let $\mathcal{F}$ be an arbitrary set family on domain $V$. Then $\mathcal{F}$ has the consecutive ones property if and only if its incompatibility graph is bipartite, and if it does not have the consecutive ones property, the incompatibility graph has an odd cycle of length at most $n + 3$.*

This theorem gives a certificate as to why a given matrix does not have COP. Similarly, the approach of testing for an ICPIA in [NS09] also gives a different certificate- a prime sub-matrix that does not have an ICPIA. Further, the above theorem can be used to check if a given matrix has COP in logspace by checking if its incompatibility graph is bipartite. [Rei84] showed that checking for bipartiteness can be done in logspace. Thus we conclude that consecutive ones testing can be done in logspace.

More recently, [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 labelling 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 3.5.16 (Theorem 4.7, [KKLV10])** *Given an interval hypergraph $\mathcal{H}$, a canonical interval labeling $l_H$ for $H$ can be computed in FL.*

We present the following reduction to see that COP testing is indeed in logspace. Given a binary matrix $M$ of order $n \times m$, let $S_i = \{j \mid M[j,i] = 1\}$. Let $\mathcal{F} = \{S_i \mid i \in [m]\}$ be this set system. Construct a hypergraph $\mathcal{H}$ with its vertex set being $\{1, 2, \ldots n\}$. The edge set of $\mathcal{H}$ is isomorphic to $\mathcal{F}$. Thus every edge in $\mathcal{H}$ represents a set in the given set system $\mathcal{F}$. Let this mapping be $\pi : E(\mathcal{H}) \to \mathcal{F}$. It is easy to see that if $M$ has COP, then $\mathcal{H}$ is an interval hypergraph. From theorem 3.5.16, it is clear that the interval labeling $l_{\mathcal{H}} : V(\mathcal{H}) \to [n]$ can be calculated in logspace. Construct sets $I_i = \{{}_{\mathcal{H}}(x) \mid x \in E, E \in E(\mathcal{H}), \pi(E) = S_i\}$, for all $i \in [m]$. Since $\mathcal{H}$ is an interval hypergraph, $I_i$ is an interval for all $i \in [m]$, and is the interval assigned to $S_i$ if $M$ has COP.

Now we have the following corollary.

**Corollary 3.5.17** *If a binary matrix $M$ has COP then the interval assignments to each* *of its columns can be calculated in FL.*

Finally, we conclude by asking about the complexity of Tree Path Assignment restricted to other subclasses of trees. In particular, is Tree Path Assignment in caterpillars easier than Tree Path assignment in general trees.

# CHAPTER 4

# Conclusion

MOVED FROM INTRO SUBSECTION IN ch:myresearch CHAPTER begin

It is an interesting fact that for a matrix with the COP, the intersection graph of the corresponding set system is an interval graph. A similar connection to a subclass of chordal graphs and a superclass of interval graphs exists for the generalization of COP. In this case, the intersection graph of the corresponding set system must be a *path graph*. Chordal graphs are of great significance, are extensively studied, and have several applications. One of the well known and interesting properties of a chordal graphs is its connection with intersection graphs [Gol04]. 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 [Ren70, Gav78, BP92]. 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]. If the chordal graph can be represented as the intersection graph of paths in a tree, then the graph is called path graph [Gol04]. Therefore, it is clear that if there is a bijection from $U$ to $V(T)$ such that for every set, the elements in it map to vertices of a unique path in $T$, then the intersection graph of the set system is indeed a path graph. However, this is only a necessary condition and can be checked efficiently because path graph recognition is polynomial time solvable[Gav78, Sch93]. Indeed, it is possible to construct a set system and tree, such that the intersection graph is a path graph, but there is no bijection between $U$ and $V(T)$ such that the sets map to paths. Path graph isomorphism is known be isomorphism-complete, see for example [KKLV10]. An interesting area of research would be to see what this result tells us about the complexity of the tree path labeling problem (not covered in this paper).

```
MOVED FROM INTRO SUBSECTION IN ch:myresearch CHAPTER
end
```

We give a characterization for 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. This thesis also discovered an exponential algorithm that computes a feasible TPL to a given hypergraph if it is a path hypergraph.

Our results have close connections to recognition of path graphs and to path graph isomorphism. Graphs which can be represented as the intersection graph of paths in a tree are called *path graphs*[Gol04]. Thus, a hypergraph $\mathcal{F}$ can be interpreted as paths in a tree, if and only if the intersection graph of $\mathcal{F}$ 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]. Path graphs are well studied in the literature [Ren70]–[Gol04]. Path graph recognition can be done in polynomial time[Gav78, Sch93]. Clearly, this is a necessary condition in terms of the intersection graph of the input hypergraph $\mathcal{F}$ in FEASIBLE TREE PATH LABELING. However, one can easily obtain a counterexample to show the insufficiency of this condition. Path graph isomorphism is known be isomorphism-complete[KKLV10]. Therefore, it is unlikely that we can solve the problem of finding feasible path labeling $\ell$ for a given $\mathcal{F}$ and tree $T$. It is definitely interesting to classify the kinds of trees and hypergraphs for which feasible path labelings can be found efficiently. These results would form a natural generalization of COP testing and interval graph isomorphism, culminating in Graph Isomorphism itself. To this effect we consider TPL on $k$-subdivided star and give a polynomial time solution for the same.

While we address the computation of TPL for a given hypergraphfrom a given target tree in this research, however optimization problems in TPL, akin to problems in Section 1.3.2 for COP, is outside the scope of this thesis. Whether TPL on general trees is solvable in $P$ remains open. So do optimization opportunities in TPL (possible extensions to optimization for COP in the Section 1.3.2.).

# REFERENCES

[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.

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

[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.

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

[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.

[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.

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

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

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

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

[HG02] Hajiaghayi and Ganjali. A note on the consecutive ones submatrix problem. *IPL: Information Processing Letters*, 83, 2002.

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

[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.

[HT02] Hochbaum and Tucker. Minimax problems with bitonic matrices. *NETWORKS: 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.

[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.

[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.

[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.

[MM96] J. Meidanis and Erasmo G. Munuera. A theory for the consecutive ones property. In *Proceedings of WSP'96 - Third South American Workshop on String Processing*, pages 194–202, 1996.

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

[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.

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

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

[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.

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