

# Generalization of the Consecutive-ones Property *fyi:* [Synopsis title is

“Generalization of Consecutive Ones Property of Binary Matrices”]

*A THESIS*

*submitted by*

ANJU SRINIVASAN

*for the award of the degree of*

MASTER OF SCIENCE *by Research*

*from the department of*

COMPUTER SCIENCE AND ENGINEERING

*at*

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Guindy, Chennai - 600036



FEBRUARY 2012



---

# CONTENTS

<b>Abstract</b>	<b>ix</b>
<b>Contents</b>	<b>xi</b>
<b>List of Tables</b>	<b>xv</b>
<b>List of Figures</b>	<b>xvii</b>
<b>Abbreviations and Notations</b>	<b>xix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Organization of the document . . . . .	1
1.2 Illustration of the problem . . . . .	2
1.2.1 Special case . . . . .	3
1.3 Basic preliminaries . . . . .	5
1.3.1 Matrices . . . . .	5
1.3.2 Sets . . . . .	7
1.3.3 Graphs . . . . .	7
1.4 Brief Survey . . . . .	8
1.4.1 Matrices with COP . . . . .	9
1.4.2 Optimization problems in COP . . . . .	10

1.5	Generalization of COP . . . . .	12
1.6	Summary of new results . . . . .	13
<b>2</b>	<b>COP – A Survey</b>	<b>15</b>
2.1	COP in Graph Theory . . . . .	16
2.2	Matrices with COP . . . . .	20
2.2.1	Tucker’s forbidden submatrices for COP . . . . .	21
2.2.2	Booth and Lueker’s <i>PQ</i> -tree – a linear COT algorithm . . . . .	22
2.2.3	<i>PQR</i> -tree – COP for set systems . . . . .	25
2.2.4	<i>PC</i> -tree– a generalization of <i>PQ</i> -tree . . . . .	29
2.2.5	ICPIA - a set cardinality based COP test . . . . .	31
2.2.6	Other COP testing algorithms . . . . .	33
2.3	Matrices without COP . . . . .	33
2.3.1	Finding forbidden submatrices . . . . .	34
2.3.2	Incompatibility graph - a certificate for no COP . . . . .	34
<b>3</b>	<b>Tree Path Labeling</b>	<b>37</b>
3.1	Summary of problems . . . . .	38
3.2	Preliminaries . . . . .	40
3.2.1	Set systems and Hypergraphs . . . . .	40
3.2.2	Path Labeling and Path Hypergraphs . . . . .	42
3.2.3	Overlap Graphs and Marginal Hyperedges . . . . .	43
3.2.4	Miscellaneous . . . . .	44
3.3	Characterization of FTPL . . . . .	44
3.4	Solution to study group accomodation problem . . . . .	52
3.5	Special target trees . . . . .	56
3.5.1	Target tree is a Path . . . . .	57
3.5.2	Target tree is a $k$ -subdivided Star . . . . .	57
3.6	TPL on arbitrary trees . . . . .	63
3.7	Complexity . . . . .	68

3.7.1	Consecutive Ones Testing is in Logspace . . . . .	68
<b>4</b>	<b>Conclusion</b>	<b>69</b>
<b>A</b>	<b>More proofs</b> <a href="#">[remove/modify]</a>	<b>73</b>
	<b>Bibliography</b>	<b>75</b>



---

# LIST OF TABLES

1.1	Students and study groups in <i>Wallace Studies Institute</i> . . . . .	2
1.2	A solution to study group accomodation problem . . . . .	3
2.1	Relationship between graph classes and graph matrices with COP or CROP. . . . .	19
2.2	A brief history of COP research . . . . .	21
2.3	Comparison of theory of $PQR$ -tree, $gPQ$ -tree, generalized $PQ$ -tree . . . . .	30





---

# LIST OF FIGURES

1.1	<i>Infinite Loop</i> street map. . . . .	4
1.2	<i>Infinite Loop</i> street map with study group routes allocated. . . . .	4
1.3	Solution to the student accommodation problem. . . . .	4
1.4	Matrices with and without COP. . . . .	6
1.5	Examples of $k$ -subdivided stars. (a) $k = 0$ (b) $k = 2$ . . . . .	14
2.1	Matrices defined in Def. 2.1.1 . . . . .	17
2.2	Tucker's forbidden subgraphs . . . . .	22
2.3	Tucker's forbidden submatrices . . . . .	23
2.4	An example for $PQ$ -tree . . . . .	24
2.5	$PC$ -tree . . . . .	31
3.1	Hypergraphs and set systems . . . . .	41
3.2	Problem solution part 1 . . . . .	53
3.3	Problem solution part 2 . . . . .	54
3.4	Problem solution part 3 . . . . .	54
3.5	Problem solution part 4 . . . . .	55
3.6	Problem solution part 5 . . . . .	55
3.7	Problem solution part 6 . . . . .	56
3.8	Problem solution part 7 . . . . .	56

3.9	(a) 8-subdivided star with 7 rays (b) 3-subdivided star with 3 rays . . . . .	57
-----	---	----



---

---

## CHAPTER 3

---

# Tree Path Labeling of Path Hypergraphs – New Results

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

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

### 3.1 Summary of Proposed Problems

In Section 1.5 we see that consecutive-ones property and its equivalent problem of interval labeling of a hypergraph is a special case of the general problem of tree path labeling of path hypergraphs. Moreover, Section 2.2.5 describes how the problem of consecutive-ones property testing can be easily seen as a simple constraint satisfaction problem involving a hypergraph or a system of sets from a universe. Every column (row) of the binary matrix can be converted into a set of non-negative integers which are the indices of the rows (columns) with 1s in that column (row). When observed in this context, if the matrix has the COP on columns (rows), a reordering of its rows (columns) will result in sets that have only consecutive integers. In other words, the sets after applying the COP row (column) permutation are intervals. In this form, one can see that this is indeed the problem of finding interval assignments to the given set system with a single permutation of the universe (set of row or column indices for COP of columns or rows, respectively) which permutes each set to an interval. The result in [NS09] (see Theorems 2.2.11 and 2.2.10) characterizes interval assignments to the sets which can be obtained from a single permutation of the universe. The cardinality of the interval assigned to it must be same as the cardinality of the set, and the intersection cardinality of any two sets must be same as the intersection cardinality of the corresponding intervals. This is a necessary and sufficient condition.

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

#### COMPUTE FEASIBLE TREE PATH LABELING

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

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

by the FEASIBLE TREE PATH LABELING problem.

### FEASIBLE TREE PATH LABELING

Input	A hypergraph $\mathcal{F}$ with vertex set $U$ , a tree $T$ , a set of paths $\mathcal{P}$ from $T$ and a bijection $\ell : \mathcal{F} \rightarrow \mathcal{P}$ .
Question	Does there exist a bijection $\phi : U \rightarrow V(T)$ 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 $S \in \mathcal{F}$ .

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

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

### COMPUTE INTERVAL LABELING

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

### COMPUTE $k$ -SUBDIVIDED STAR PATH LABELING

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

An algorithm for COMPUTE FEASIBLE TREE PATH LABELING on general trees which is less efficient than polynomial time is presented in Section 3.6.

## 3.2 Preliminaries to new results

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

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

### 3.2.1 Set systems and Hypergraphs

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

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

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

A set system  $\mathcal{F}$  can also be visualized as a *hypergraph*  $\mathcal{F}_H$  whose vertex set is  $U$  and hyperedges are the sets in  $\mathcal{F}$ . This is a known representation for interval systems in literature [BLS99, KKL10]. We extend this definition here to path systems. Due to the equivalence of set system and hypergraph as far as this thesis is concerned, we drop the subscript  $H$  in the notation and refer to both the structures by  $\mathcal{F}$ . Thus hypergraphs and set systems are synonyms. So are hyperedges and sets.

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

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

Figure 3.1 shows an example of hypergraphs and hypergraph isomorphism.

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

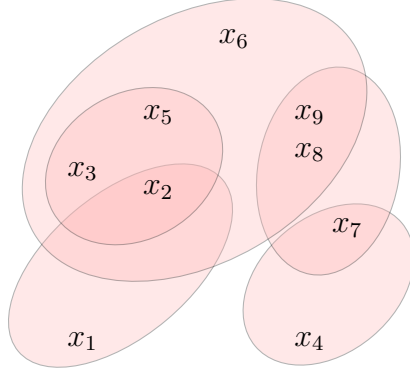
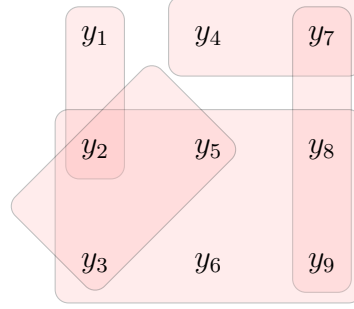
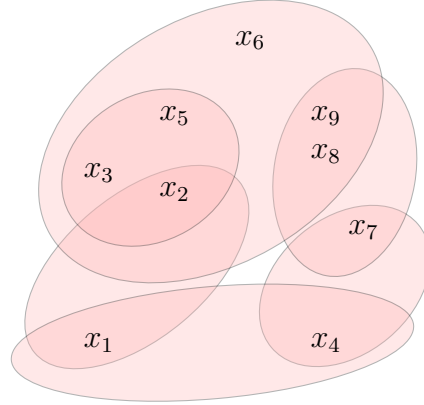
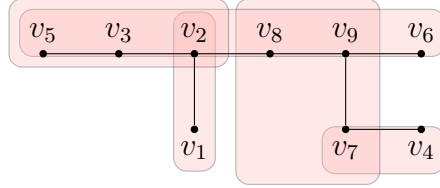
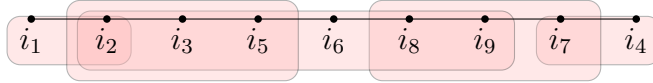
 $\mathcal{F}_1 :$  $\mathcal{F}_2 :$  $\mathcal{F}_3 :$  $\mathcal{P} :$  $\mathcal{I} :$ 

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



### 3.2.2 Path Labeling and Path Hypergraphs

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

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

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

**Definition 3.2.2 (Path labeling of a hypergraph).** Consider a hypergraph  $\mathcal{F}$  and a path system  $\mathcal{P}$  from tree  $T$  such that  $\mathbb{I}(\mathcal{F}) \cong \mathbb{I}(\mathcal{P})$ . Then the associated bijection  $\ell : \mathcal{F} \rightarrow \mathcal{P}$  due to this isomorphism is called a *path labeling* of  $\mathcal{F}$  from  $T$ . The path system  $\mathcal{P}$  is alternatively denoted as  $\mathcal{F}^\ell$ . The path labeling may also be denoted by ordered pairs  $(\mathcal{F}, \ell)$  or  $(\mathcal{F}, \ell, T)$ .

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

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

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

all  $S \in \mathcal{F}$ .

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

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

- $\mathcal{F} \cong \mathcal{F}^\ell$
  - if the above hypergraph isomorphism is  $\phi : \text{supp}(\mathcal{F}) \rightarrow \text{supp}(\mathcal{F}^\ell)$ , then  $p_\phi = \ell$
- This isomorphism  $\phi$  is called the **feasibility hypergraph isomorphism** or feasibility isomorphism.

One of the problems solved in this thesis FEASIBLE TREE PATH LABELING characterizes feasible tree path labeling.

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

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

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

### 3.2.3 Overlap Graphs and Marginal Hyperedges

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

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

A hyperedge  $S \in \mathcal{F}$  is called *marginal* if for all  $S' \bowtie S$ , the overlaps  $S \cap S'$  form a single inclusion chain [KKLV10] (see Definition 1.3.7). Additionally, if  $S$  is such that it is contained in no other marginal hyperedge in  $\mathcal{F}$ , then it is called *super-marginal*.

### 3.2.4 Miscellaneous

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

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

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

In-trees are used in Section 3.6 which describes our (albeit non-polynomial time) solution to COMPUTE FEASIBLE TREE PATH LABELING problem.

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

## 3.3 Characterization of Feasible Tree Path Labeling

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

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

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

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

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

In the remaining part of this section we show that  $(\mathcal{F}, \ell)$  is feasible if and only if it is an ICPPL. Algorithm 3 takes  $(\mathcal{F}, \ell)$  as input and computes the feasibility bijection if it is feasible or else reports failure. This algorithm recursively does two levels of filtering of  $(\mathcal{F}, \ell)$  to make it simpler in terms of set intersections while retaining a non-empty subset of isomorphisms, if any, between  $\mathcal{F}$  and  $\mathcal{F}'$ . These two filters are `filter common leaf` in Algorithm 1 and `filter fix leaf` in Algorithm 2.

First, we present Algorithm 1 which describes `filter common leaf`, and prove its correctness. This algorithm refines the path labeling by processing pairs of paths in  $\mathcal{F}'$  that share a leaf until no two paths in the new path labeling share any leaf.

---

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

---

```

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
6:    $\ell_j(S_1 \cap S_2) \leftarrow \ell_{j-1}(S_1) \cap \ell_{j-1}(S_2)$ 
      | Carry forward the path
      | labeling for all existing sets
      | other than  $S_1, S_2$ 
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:    $j \leftarrow j + 1$ 
10: end while
11:  $\mathcal{F}' \leftarrow \mathcal{F}_j$ ,  $\ell' \leftarrow \ell_j$ 
12: return  $(\mathcal{F}', \ell')$ 

```

---

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

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

tree as a path i.e. it is an interval labeling, and (ii) it preserves pairwise intersection cardinalities i.e. it is an ICPIA, then it is a feasible path labeling.

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

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

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

**Lemma 3.3.3.** *If  $\ell$  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,  $|S_1 \cap S_2| - |S_1 \cap S_2 \cap S_3| = |P_1 \cap P_2| - |P_1 \cap P_2 \cap P_3| = |(P_1 \cap P_2) \setminus P_3| = |P_1 \cap (P_2 \setminus P_3)|$ .  $\square$

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

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

*Proof.* Consider the sets  $S_2 \cap S_1$ ,  $S_3 \cap S_1$ , and  $S_4 \cap S_1$ , and their images  $P_2 \cap P_1$ ,  $P_3 \cap P_1$ , and  $P_4 \cap P_1$  respectively. It is clear that this path labeling preserves pairwise intersection cardinalities. In this case, it is clear that the 3 sets are assigned paths in the path  $P_1$ . Now by applying Lemma 3.3.1, it follows that  $|\cap_{i=1}^4 S_i| = |\cap_{i=1}^4 P_i|$ .  $\square$

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

$$\begin{aligned} |(S_1 \setminus S_2) \cap S_3 \cap S_4| &= |(P_1 \setminus P_2) \cap P_3 \cap P_4| \\ |(S_2 \setminus S_1) \cap S_3 \cap S_4| &= |(P_2 \setminus P_1) \cap P_3 \cap P_4| \end{aligned}$$

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

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

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

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

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

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

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

- I  $\ell_j(R)$  is a path in  $T$ , for all  $R \in \mathcal{F}_j$
- II  $|R| = |\ell_j(R)|$ , 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 1,  $S_1 \cap S_2, S_1 \setminus S_2, S_2 \setminus S_1$  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  $\ell_j$  in terms of  $\ell_{j-1}$ , 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 3.3.3 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*

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

□

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

*Proof.* Since  $(\mathcal{F}, \ell)$  is feasible, by Lemma 3.3.7  $(\mathcal{F}_j, \ell_j)$  for every iteration  $j > 0$  is feasible. Also, every hypergraph isomorphism  $\phi : \text{supp}(\mathcal{F}) \rightarrow V(T)$  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  $x \in \text{supp}(\mathcal{F})$ , for all  $v \in V(T)$ , if  $(x, v) \in \phi$  then  $v \in \ell_j(S)$  for all  $S \in \mathcal{F}_j$  such that  $x \in S$ . In other words, `filter common leaf` outputs a filtered path labeling that “preserves” hypergraph isomorphisms of the original path labeling.

□

As a result of `filter common leaf` each leaf  $v$  in  $T$  is such that there is exactly one set in  $\mathcal{F}$  with  $v$  as a vertex in the path assigned to it. `filter fix leaf`, in Algorithm 2, identifies elements in  $\text{supp}(\mathcal{F})$  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  $\ell_{\text{new}}$  such that  $\ell_{\text{new}}(\{x\}) = \{v\}$  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  $\ell_{\text{new}}$  except leaf labels (a path that has only a leaf is called the *leaf label* for the corresponding single element hyperedge or set) are paths from a new pruned tree  $T_0 = T \setminus \{v \mid v \text{ is a leaf in } T\}$ . Algorithm 2 is now presented with details.



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


---

```

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

```

---

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

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

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

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

*Proof.* Following Lemma 3.3.10, we know that set  $X$  will not be empty in any iteration of the **while** loop if input ICPPL  $(\mathcal{F}, \ell)$  is feasible and thus  $\ell_j$  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  $x$  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.  $\square$

We have seen two filtering algorithms above, namely, Algorithm 1 for `filter common`

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

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

Since their pre-images have been found, 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  $\mathcal{F}_2$ . It is now clear that we have a subproblem with a new hypergraph  $\mathcal{F}'$ , new tree path labeling  $\ell'$  and target tree  $T'$ . The tree pruning algorithm is recursively called on  $\mathcal{F}', \ell', T'$ . The recursive call returns the bijection  $\phi''$  for the rest of the elements in  $\text{supp}(\mathcal{F})$  which along with the leaf labels  $\phi'$  computed earlier gives us a hypergraph isomorphism  $\phi$  for the input  $\mathcal{F}, \ell, T$ . Lemma 3.3.12 proves the correctness of this computation.

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

*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 input target tree  $T$  (of the current recursive call). Moreover, by Lemma 3.3.9 and Lemma 3.3.10 it is consistent with the input tree path labeling  $\ell$  (of the current recursive call). The tree pruning is done by only removing leaves in each call to the function and is done till the tree becomes empty. Thus the returned

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

```

---

function  $\phi : \text{supp}(\mathcal{F}) \rightarrow V(T)$  is a union of mutually exclusive one-to-one functions exhausting all vertices of the tree. In other words, it is a bijection from  $\text{supp}(\mathcal{F})$  to  $V(T)$  inducing the given path labeling  $\ell$  and thus a hypergraph isomorphism.  $\square$

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

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

*Proof.* From Lemma 3.3.12, we know that if  $(\mathcal{F}, \ell)$  is an ICPPL and if Algorithm 3 with  $(\mathcal{F}, \ell, T)$  as input returns a non-empty function, then  $(\mathcal{F}, \ell)$  is feasible. Now consider the case where  $(\mathcal{F}, \ell)$  is feasible, i.e. there exists a hypergraph isomorphism  $\phi$  such that  $\ell_\phi = \ell$ . Lemma 3.3.9 and Lemma 3.3.10 show us that filter common leaf and filter fix leaf do not exit if input is feasible. Thus Algorithm 3 returns a non-empty function.  $\square$

## 3.4 Solution to study group accomodation problem

In this section we run through the filtering and tree pruning algorithm seen in Section 3.3 on an example. The example problem is the study group accomodation problem

posed in Section 1.2. The solution is depicted in Figure 3.2, Figure 3.3, Figure 3.4, Figure 3.5, Figure 3.6, Figure 3.7, Figure 3.8 and the figures along with their captions are self-explanatory.

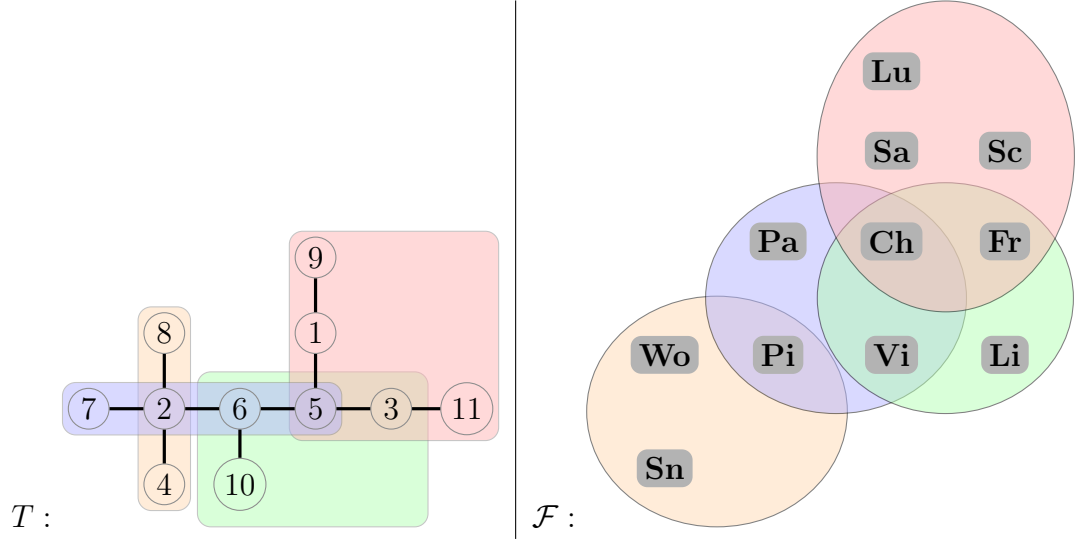


Figure 3.2: The problem is shown again here using the input described in Section 1.2. The target tree is  $T$  with vertex set  $\{1, 2, \dots, 11\}$  on the left hand side along with the path system (hypergraph) –  $\{7, 2, 6, 5\}, \{8, 2, 4\}, \{10, 6, 5, 3\}, \{9, 1, 5, 3, 11\}$ . The hypergraph is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

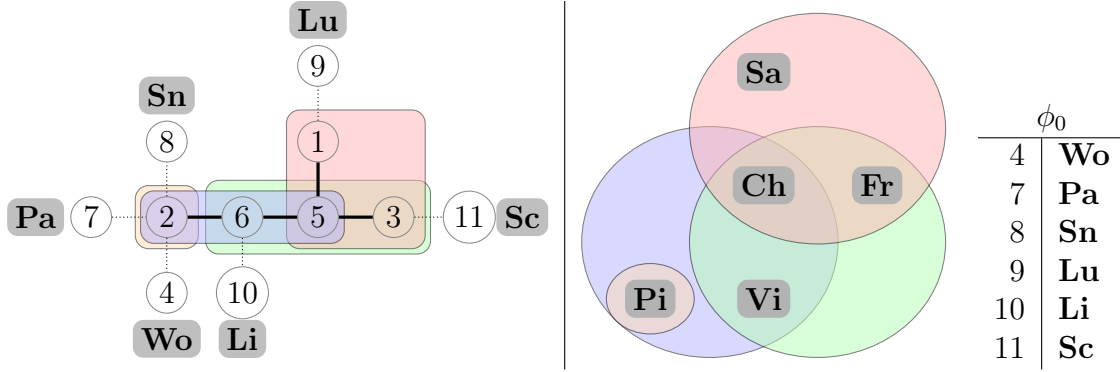


Figure 3.3: It can be observed in Figure 3.2, that none of the leaves are shared by more than one path. Hence, `filter common leaf` does not alter the hypergraphs. `filter fix leaf` finds the leaf assignments as shown – elements **Pa**, **Sn**, **Lu**, **Sc**, **Li**, **Wo** are assigned to vertices 7, 8, 9, 11, 10, 4 respectively. Computation of these leaf labels are deterministic except for **Lu**, **Sc** where the algorithm arbitrarily choses them from  $X = \{\text{Lu}, \text{Sa}, \text{Sc}\}$  and for **Sn**, **Wo** arbitrarily choses from  $X = \{\text{Sn}, \text{Wo}\}$ . Here we show only one such choice and clearly one could have chosen another leaf label (e.g. **Sa** on 9 or **Wo** on 8) to get a different feasible hypergraph isomorphism. After this, the leaves of  $T$  are pruned and the hypergraph updated accordingly by `get-hypergraph-isomorphism` subroutine – the pruned leaves are shown with dotted edges and the corresponding Venn diagrams show the updated hypergraphs.  $\phi_0$  is this leaf assignment and the algorithm proceeds to solve the subproblem with the new hypergraphs and path labeling which is:  $\{2, 6, 5\}$ ,  $\{2\}$ ,  $\{6, 5, 3\}$ ,  $\{1, 5, 3\}$  labeled with  $\{\text{Pi}, \text{Vi}, \text{Ch}\}$ ,  $\{\text{Pi}\}$ ,  $\{\text{Ch}, \text{Fr}, \text{Vi}\}$ ,  $\{\text{Sa}, \text{Ch}, \text{Fr}\}$  respectively.

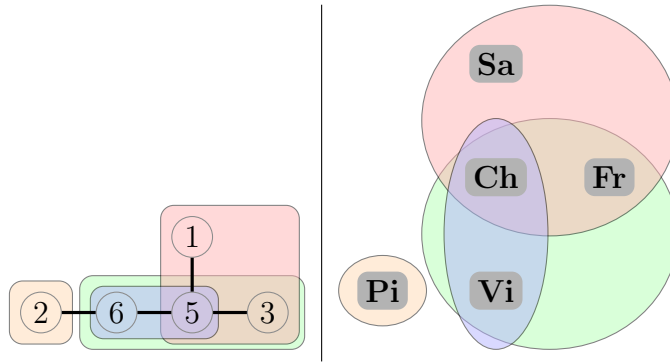


Figure 3.4: In the subproblem shown in Figure 3.3, it can be noted that there are two leaves which are shared by more than one paths – 2 and 3. Hence `filter common leaf` filters this out one by one. Here we show the handling of leaf 2 which was shared by paths  $\{2, 6, 5\}$  ( $= P_1$ , say) and  $\{2\}$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

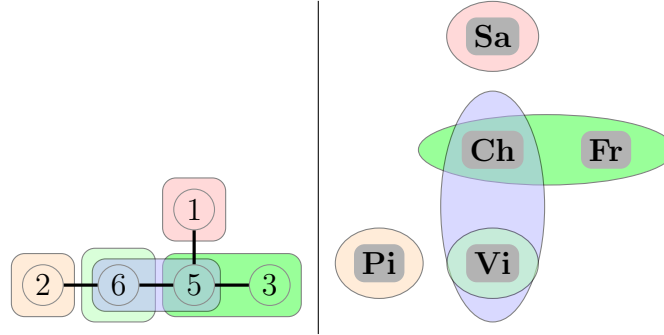


Figure 3.5: After handling leaf 2, `filter common leaf` proceeds to handle leaf 3 which is shared by paths  $\{1, 5, 3\}$  and  $\{6, 5, 3\}$ . `filter common leaf` removes these two paths and adds their intersection and set differences –  $\{5, 3\}$ ,  $\{1\}$  and  $\{6\}$ . Correspondingly, hyperedges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

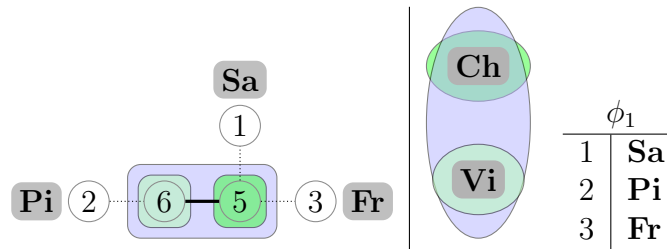


Figure 3.6: `filter fix leaf` computes the leaf assignment  $\phi_1 - \mathbf{Pi}, \mathbf{Sa}, \mathbf{Fr}$  on 2,1,3 respectively. These assignments are deterministic since  $X$  consisted of only single elements for each of these leaves. After this the rest of the labeling is fairly obvious but we present it in Figure 3.7 for the sake of completeness.

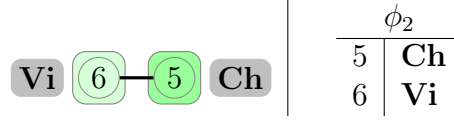


Figure 3.7: The last part of the isomorphism is computed as the leaf label  $\phi_2$  which maps **Ch**, **Vi** to 5, 6 respectively. Figure 3.8 shows the final feasibility isomorphism which is the union of all the leaf labels.

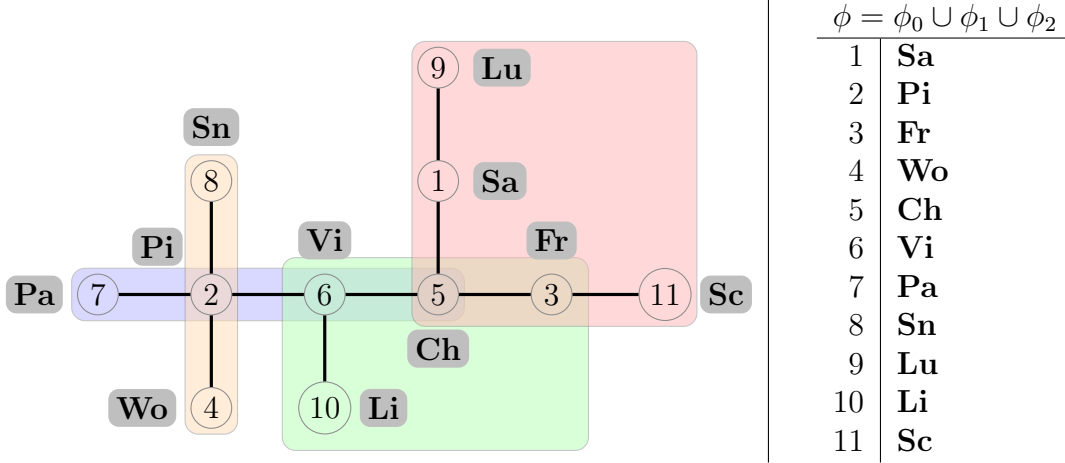


Figure 3.8: The solution to the problem presented in Figure 3.2. The feasibility hypergraph isomorphism  $\phi$  is computed as the union of the leaf labels  $\phi_0, \phi_1, \phi_2$ . One can easily verify that this isomorphism does indeed induce the path labeling given in the original input.

### 3.5 Computing feasible TPL with special target trees

Section 3.3 described properties that a TPL must have for it to be feasible. The next problem of interest is to test if a given hypergraph is a path hypergraph with respect to a given target tree<sup>1</sup>. In other words, the problem is to find out if a feasible tree path labeling exists from a given target tree for a given hypergraph. In this section we will see two special cases of this problem where the target tree is from a particular family of trees. The first one, where the tree is a path as described by the COMPUTE INTERVAL LABELING problem (see Section 3.1) is shown to be equivalent to the well studied problem of consecutive-ones in Section 3.5.1. The second one, where the tree is a  $k$ -subdivided tree as described by COMPUTE  $k$ -SUBDIVIDED STAR PATH LABELING (see Section 3.1), has been solved using a polynomial time algorithm. The latter problem enforces some conditions on the hypergraph too which will be seen in Section 3.5.2.

1. A larger problem would be to find if a given hypergraph is a path hypergraph with respect to any tree. This problem is not addressed in this thesis.

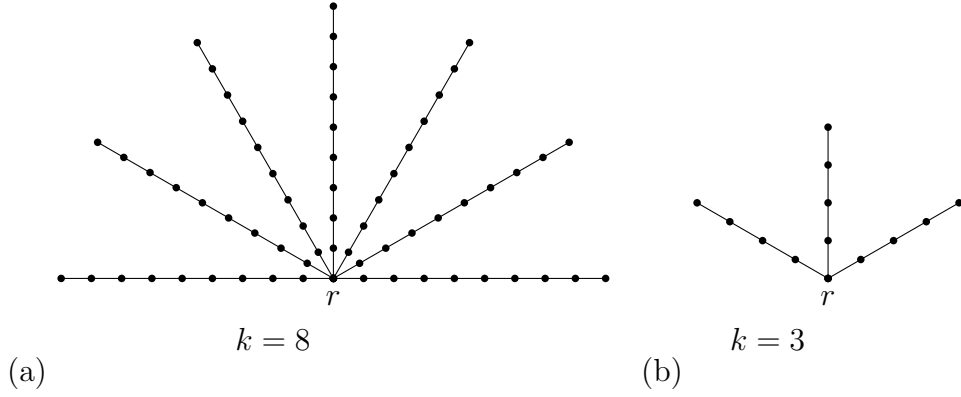


Figure 3.9: (a) 8-subdivided star with 7 rays (b) 3-subdivided star with 3 rays

### 3.5.1 Target tree is a Path

In Section 1.5 it is described how COP is a special case of TPL as described by the COMPUTE INTERVAL LABELING problem. In this section we will dwell on this in more detail.

*dire:* The special case of TPL where the target tree  $T$  is a path. Hence, all path labels are can be viewed as intervals assigned to the sets in  $\mathcal{F}$ . *[needs work]*

It is shown, in [NS09], that the filtering algorithms outlined above need only preserve pairwise intersection cardinalities, and higher level intersection cardinalities are preserved by the Helly Property of intervals. 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.

*dire:* [ELABORATE]

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

In this section we consider the problem of assigning paths from a  $k$ -subdivided star  $T$  to a given set system  $\mathcal{F}$  as described by the COMPUTE  $k$ -SUBDIVIDED STAR PATH LABELING problem. We consider  $\mathcal{F}$  for which the overlap graph  $\mathbb{O}(\mathcal{F})$  is connected. The overlap graph is well-known from the work of [KKLV10, NS09, Hsu02]. We use the notation in [KKLV10]. Recall from Section 3.2 that hyperedges  $S$  and  $S'$  are said to overlap, denoted by  $S \bowtie S'$ , 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  $R_1, \dots, R_p$ , and the number of vertices in  $R_i$ ,  $i \in [p]$  is  $k + 2$ . Let  $\langle v_{i1}, \dots, v_{i(k+2)} = r \rangle$  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.

*dire:* [REPEAT!!]

In this section we consider the problem of assigning paths from a  $k$ -subdivided star  $T$  to a given set system  $\mathcal{F}$  such that each set  $X \in \mathcal{F}$  is of cardinality at most  $k + 2$ . Secondly, we present our results only for the case when overlap graph  $\mathbb{O}(\mathcal{F})$  is connected. 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. Recall that we also consider the special case when all hyperedges are of cardinality at most  $k+2$ . By definition, 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  $R_1, \dots, R_p$ , and the number of vertices in  $R_i$ ,  $i \in [p]$  is  $k+2$ . Let  $\langle v_{i1}, \dots, v_{i(k+2)} = r \rangle$  denote the sequence of vertices in  $R_i$ , where  $v_{i1}$  is the leaf. Note that  $r$  is a common vertex to all  $R_i$ . ]

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

For each hyperedge  $X \in \mathcal{O}$ , we will maintain a 2-tuple of non-negative numbers  $\langle p_1(X), p_2(X) \rangle$ . The numbers satisfy the property that  $p_1(X) + p_2(X) \leq |X|$ , and at the end of path labeling, for each  $X$ ,  $p_1(X) + p_2(X) = |X|$ . This signifies the algorithm tracking the lengths of subpaths of the path assigned to  $X$  from at most two rays. We also maintain another parameter called the *residue* of  $X$  denoted by  $s(X) = |X| - p_1(X)$ . This signifies the residue path length that must be assigned to  $X$  which must be from another ray. For instance, if  $X$  is labeled a path from only one ray, then  $p_1(X) = |X|$ ,  $p_2(X) = 0$  and  $s(X) = 0$ .

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

$\mathcal{L}_1^i$  *Labeled without  $r$ .* Those that have been labeled with a path which does not contain  $r$  in one of the previous iterations.

$$\mathcal{L}_1^i = \{X \mid p_1(X) = |X| \text{ and } p_2(X) = 0 \text{ and } s(X) = 0, X \in \mathcal{O}\}$$

$\mathcal{L}_2^i$  *Labeled with  $r$ .* Those that have been labeled with two subpaths of  $\ell(X)$  containing  $r$  from two different rays in two previous iterations.

$$\mathcal{L}_2^i = \{X \mid 0 < p_1(X), p_2(X) < |X| = p_1(X) + p_2(X) \text{ and } s(X) = 0, X \in \mathcal{O}\}$$

$\mathcal{T}_1^i$  *Type 1 / partially labeled.* Those that have been labeled with one path containing  $r$  from a single ray in one of the previous iterations. Here,  $p_1(X)$  denotes the length

of the subpath of  $\ell(X)$  that  $X$  has been so far labeled with.

$$\mathcal{T}_1^i = \{X \mid 0 < p_1(X) < |X| \text{ and } p_2(X) = 0 \text{ and } s(X) = |X| - p_1(X), X \in \mathcal{O}\}$$

$\mathcal{T}_2^i$  *Type 2 / not labeled.* Those that have not been labeled with a path in any previous iteration.

$$\mathcal{T}_2^i = \{X \mid p_1(X) = p_2(X) = 0 \text{ and } s(X) = |X|, X \in \mathcal{O}\}$$

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

I. **Iteration 1:** Let  $S = \{X_1, \dots, X_s\}$  denote the super-marginal hyperedges from  $\mathcal{O}_1$ .

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

II. **Iteration  $i$ :** Let  $X$  be a hyperedge from  $\mathcal{O}_i$  such that there exists  $Y \in \mathcal{L}_1^i \cup \mathcal{L}_2^i$  and  $X \not\sim Y$ . Further let  $Z \in \mathcal{L}_1^i \cup \mathcal{L}_2^i$  such that  $Z \not\sim Y$ . If  $X \in \mathcal{T}_2^i$ , and if there are multiple  $Y$  candidates then any  $Y$  is selected. On the other hand, if  $X \in \mathcal{T}_1^i$ , then  $X$  has a partial path assignment,  $\ell'(X)$  from a previous iteration, say from ray  $R_j$ . Then,  $Y$  is chosen such that  $X \cap Y$  has a non-empty intersection with a ray different from  $R_j$ . The key things that are done in assigning a path to  $X$  are as follows. The *end* of path  $\ell(Y)$  where  $\ell(X)$  would overlap is found, and then based on this the existence of a feasible assignment is decided. It is important to note that since  $X \not\sim Y$ ,  $\ell(X) \not\sim \ell(Y)$  in any feasible assignment. Therefore, the notion of the *end* at which  $\ell(X)$  and  $\ell(Y)$  overlap is unambiguous, since for any path, there are two end points.

- (a) *End point of  $\ell(Y)$  where  $\ell(X)$  overlaps depends on  $X \cap Z$ :* If  $X \cap Z \neq \emptyset$ , then  $\ell(X)$  has an overlap of  $|X \cap Y|$  at that end of  $\ell(Y)$  at which  $\ell(Y)$  and  $\ell(Z)$  overlap. If  $X \cap Z = \emptyset$ , then  $\ell(X)$  has an overlap of  $|X \cap Y|$  at that end of  $\ell(Y)$  where  $\ell(Y)$  and  $\ell(Z)$  do not intersect.
- (b) *Any path of length  $s(X)$  at the appropriate end contains  $r$ :* If  $X \in \mathcal{T}_1^i$  then after finding the appropriate end as in step IIa this the unique path of length  $s(X)$  should end at  $r$ . If not, we exit reporting failure. Else,  $\ell(X)$  is computed as union of  $\ell'(X)$  and this path. If any three-way intersection cardinality is violated with this new assignment, then exit, reporting failure. Otherwise,  $X$  is added to  $\mathcal{L}_2^{i+1}$ . On the other hand, if  $X \in \mathcal{T}_2^i$ , then after step IIa,  $\ell(X)$  or  $\ell'(X)$

is unique up to the root and including it. Clearly, the vertices  $\ell(X)$  or  $\ell'(X)$  contains depends on  $|X|$  and  $|X \cap Y|$ . If any three way intersection cardinality is violated due to this assignment, exit, reporting failure. Otherwise,  $p_1(X)$  is updated as the length of the assigned path, and  $s(X) = |X| - p_1(X)$ . If  $s(X) > 0$ , then  $X$  is added to  $\mathcal{T}_1^{i+1}$ . If  $s(X) = 0$ , then  $X$  is added to  $\mathcal{L}_1^{i+1}$ .

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

**Proof of Correctness and Analysis of Running Time:** It is clear that the algorithm runs in polynomial time, as at each step, at most three-way intersection cardinalities need to be checked. Further, finding super-marginal hyperedges can also be done in polynomial time, as it involves considering the overlap regions and checking if the inclusion partial order contains a single minimal element. In particular, once the super-marginal edges are identified, each iteration involves finding the next hyperedge to consider, and testing for a path to that hyperedge. To identify the next hyperedge to consider, we consider the breadth first layering of the hyperedges with the zeroeth layer consisting of the super-marginal hyperedges. Since  $\mathcal{O}$  is connected, it follows that all hyperedges of  $\mathcal{O}$  will be considered by the algorithm. Once a hyperedge is considered, the path to be assigned to it can also be computed in constant time. In particular, in the algorithm the path to be assigned to  $X$  depends on  $\ell(Y), \ell(Z), s(X)$  and the presence or absence of  $r$  in the candidate partial path  $\ell'(X)$ . Therefore, once the super-marginal edges are identified, the running time of the algorithm is linear in the size of the input. By the technique used for constructing prime matrices [Hsu02], the super-marginal edges can be found in linear time in the input size. Therefore, the algorithm can be implemented to run in linear time in the input size.

The proof of correctness uses the following main properties:

1. The  $k$ -subdivided star has a very symmetric structure. This symmetry is quantified based on the following observation – either there are no feasible path labelings of  $\mathcal{O}$  using paths from  $T$ , or there are exactly  $p!$  feasible path labelings. In other words, there is either no feasible assignment, or effectively a unique assignment modulo symmetry.
2. The  $p$  super-marginal hyperedges, if they exist, will each be assigned a path from distinct rays, and each such path contains the leaf.
3. For a candidate hyperedge  $X$ , the partial path assignment  $\ell'(X)$  is decided by its

overlap with  $\ell(Y)$  and cardinality of intersection with  $\ell(Z)$ .

These properties are formalized as follows:

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

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

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

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

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

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

These lemmas now are used to prove the following theorem.

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

*Proof. Outline.* If the algorithm outputs a path labeling  $\ell$ , then it is clear that it is an ICPPL. The reason is that the algorithm checks that three-way intersection cardinalities

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

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

## 3.6 TPL on arbitrary trees

In this section we describe an algorithm for the COMPUTE FEASIBLE TREE PATH LABELING problem where the target tree can be any arbitrary tree. In essence, we provide a characterization for path hypergraphs (see Section 3.2.2 for definition). We first study some properties of the overlap components of the input hypergraph and introduce some theory in that area. This is in the same vein as the theory used in [Hsu02, NS09] but we extend it to TPL. We use this theory to decompose COMPUTE FEASIBLE TREE PATH LABELING into subproblems. Each subproblem is on a sub-hypergraph of the input, in which for each hyperedge there is another hyperedge in the sub-hypergraph that overlaps with it. These sub-hypergraphs are called overlap components (see Section 3.2.3 for definition). As a consequence of this characterization, aside from the overlap component subproblems, we identify two other subproblems that must be solved to obtain an ICPPL. The inefficiency of our algorithm in terms of polynomial time solvability comes from these two subproblems. We leave these subproblems open to be solved in P and only mention the obvious brute force method of solution.

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

Consider the overlap graph and overlap components of  $\mathbb{O}(\mathcal{F})$  defined in Section 3.2.3. We use this to decompose  $M$  as described in [Hsu02, NS09]. A *prime submatrix* of  $M$  is defined as the matrix formed by a set of columns of  $M$  which correspond to an overlap component. Let us denote the prime submatrices of  $M$  by  $M_1, \dots, M_p$ , each corresponding to one of the  $p$  overlap components of  $\mathcal{F}$ . Clearly, two distinct prime submatrices have mutually exclusive sets of columns of  $M$ . Let  $col(M_i)$  be the set of columns in the submatrix  $M_i$ . The support of a prime submatrix  $M_i$  and support of a set of prime submatrices  $X$  are defined as follows. Note that for each  $i$ ,  $supp(M_i) \subseteq U$  where  $U$  is the vertex set of hypergraph  $\mathcal{F}$ .

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

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

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

This relation is the same as that defined in [NS09]. The prime submatrices and the above relation can be defined for any hypergraph since an overlap graph is defined for any hypergraph. We will use this structure of prime submatrices to present our results on an ICPPL for a hypergraph  $\mathcal{F}$ . Now we present a few lemma and a theorem that show that  $\preceq$  is a partial order. An alternate notation for  $(M_i, M_j) \in \preceq$  is  $M_i \preceq M_j$ .

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

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

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

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

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

- i.  $\preceq$  is a partial order on  $\mathbb{P}$ .
- ii.  $\preceq$  uniquely partitions  $\mathbb{P}$  into  $\mathbb{P}_1, \mathbb{P}_2, \dots, \mathbb{P}_r$  for some  $r \geq 1$ .
- iii. If  $M$  has COP, then for each  $i \in [r]$ ,  $\preceq$  induces a total order in  $\mathbb{P}_i$ .

Since we are concerned with the generalization of COP, we generalize the condition (iii) in Theorem 3.6.5 from total order to in-trees as follows. The in-tree is derived from the Hasse diagram of  $\preceq$ . Each of these in-trees will be on disjoint vertex sets, which in this case would be disjoint sets of prime-submatrices  $\mathbb{P}_i$ ,  $i \in [r]$  for some  $r > 0$ . For each  $\mathbb{P}_i$ , the in-tree is rooted at a maximal upper bound under  $\preceq$ . The in-trees are specified by selecting the appropriate edges from the Hasse diagram associated with  $\preceq$ . This is defined in Theorem 3.6.6 before which we see the covering relation for  $\preceq$  defined below.

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

It is easy to see that  $\preceq_c$  is the covering relation of the partial order  $\preceq$ .

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

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



root is a maximal upper bound in  $\preceq$ . □

Let the partition of  $X$  given by Theorem 3.6.6 be  $\{X_1, \dots, X_r\}$ . 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 3.6.7.** *Let  $M$  be a matrix and let  $X$  be the directed graph whose vertices are in correspondence with the prime submatrices of  $M$ . Further let  $\{X_1, \dots, X_r\}$  be the partition of  $X$  into in-trees as defined above. Then, matrix  $M$  has an ICPPL in tree  $T$  if and only if  $T$  can be partitioned into vertex disjoint subtrees  $\{T_1, T_2, \dots, T_r\}$  such that, for each  $1 \leq i \leq r$ , the set of prime submatrices corresponding to vertices in  $X_i$  has an ICPPL in  $T_i$ .*

*Proof.* Let us consider the reverse direction first. Let us assume that  $T$  can be partitioned into  $T_1, \dots, T_r$  such that for each  $1 \leq i \leq r$ , the set of prime submatrices corresponding to vertices in  $X_i$  has an ICPPL in  $T_i$ . It is clear from the properties of the partial order  $\preceq$  that these ICPPLs naturally yield an ICPPL of  $M$  in  $T$ . The main property used in this inference is that for each  $1 \leq i \neq j \leq r$ ,  $\text{supp}(X_i) \cap \text{supp}(X_j) = \phi$ .

To prove the forward direction, we show that if  $M$  has an ICPPL, say  $\mathcal{A}$ , in  $T$ , then there exists a partition of  $T$  into vertex disjoint subtree  $T_1, \dots, T_r$  such that for each  $1 \leq i \leq r$ , the set of prime submatrices corresponding to vertices in  $X_i$  has an ICPPL 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 submatrix in  $X_{i,0}$ . Consider the paths assigned under  $\mathcal{A}$  to the sets in the prime submatrix 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 ICPPL, and the support of other prime submatrices in  $X_i$  are contained in the support of the matrix in  $X_{i,0}$ . Secondly, for each  $1 \leq i \neq j \leq r$ ,  $\text{supp}(X_i) \cap \text{supp}(X_j) = \phi$ , and since  $\mathcal{A}$  is an ICPPL, it follows that  $T_i$  and  $T_j$  are vertex disjoint. Finally, since  $|U| = |V(T)|$ , it follows that  $T_1, \dots, T_r$  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 ICPPL in  $T_i$ . Hence the lemma. □

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

**Lemma 3.6.8.** *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  $\{X_1, \dots, X_r\}$  be the partition of  $X$  into in-trees as defined earlier in this section. Let  $T$  be the given tree and let  $\{T_1, \dots, T_r\}$  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 ICPPL in  $T_i$  if and only if the matrix in  $X_{i,0}$  has an ICPPL in  $T_i$  and all other matrices in  $X_i$  have an **ICPIA**.*

*Proof.* The proof is based on the following fact-  $\preceq$  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 corresponding submatrices. Therefore, any ICPPL to a prime submatrix that is not the root is contained in a path assigned to the sets in the parent matrix. Consequently, any ICPPL to the prime submatrix that is not at the root is an ICPIA, and any ICPIA can be used to construct an ICPPL to the matrices corresponding to nodes in  $X_i$  provided the matrix in the root has an ICPPL in  $T_i$ . Hence the lemma.  $\square$

Lemma 3.6.7 and Lemma 3.6.8 point out two algorithmic challenges in finding an ICPPL for a given set system  $\mathcal{F}$  in a tree  $T$ . Given  $\mathcal{F}$ , finding  $X$  and its partition  $\{X_1, \dots, X_r\}$  into in-trees can be done in polynomial time. On the other hand, as per lemma 3.6.7 we need to partition  $T$  into vertex disjoint sub-trees  $\{T_1, \dots, T_r\}$  such that for each  $i$ , the set of matrices corresponding to nodes in  $X_i$  have an ICPPL 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 3.6.8 which is to assign an ICPPL from a given tree to the matrix associated with the root node of  $X_i$ .

---

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

---

Identify the prime submatrices. This is done by constructing the strict overlap graph and identify connected components. Each connected component yields a prime submatrix.

Construct the partial order  $\preceq$  on the set of prime submatrices.

Construct the partition  $X_1, \dots, X_r$  of the prime submatrices induced by  $\preceq$

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  $T_1, \dots, T_r$  such that matrices in  $X_{i,0}$  has an ICPPL in  $T_i$ . If not such partition exists, exit with negative answer.

---

## 3.7 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 set in  $\mathcal{F}$  is a path in  $T$ .

*dire:* [it is known that if the given tree is a path a feasible assignment can be found in polynomial time, and we observe that it can actually be done in logspace. ]

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.

### 3.7.1 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 if and only if 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 3.7.1 (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 submatrix 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 3.7.2 (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  $\{1, 2, \dots, n\}$ . 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  $\pi : E(\mathcal{H}) \rightarrow \mathcal{F}$ . It is easy to see that if  $M$  has COP, then  $\mathcal{H}$  is an interval hypergraph. From theorem 3.7.2, it is clear that the interval labeling  $l_{\mathcal{H}} : V(\mathcal{H}) \rightarrow [n]$  can be calculated in logspace. Construct sets  $I_i = \{l_{\mathcal{H}}(x) \mid x \in E, E \in E(\mathcal{H}), \pi(E) = S_i\}$ , 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 3.7.3.** *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.

---

---

# APPENDIX A

---

## More proofs [\[remove/modify\]](#)

**Lemma A.0.4.** *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.  $Q \cap (Q_1 \setminus Q_2) = \emptyset$ . Then  $Q \cap Q_1 \cap Q_2 = Q \cap Q_2$ . Similarly, if  $Q \cap (Q_2 \setminus Q_1) = \emptyset$ ,  $Q \cap Q_1 \cap Q_2 = Q \cap Q_1$ . 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  $Q \cap (Q_1 \setminus Q_2) \neq \emptyset$  and  $Q \cap (Q_2 \setminus Q_1) \neq \emptyset$ , 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 □