Generalization of the Consecutive-ones Property

A THESIS

submitted by

ANJU SRINIVASAN

for the award of the degree of

MASTER OF SCIENCE *by Research*

from the department of

COMPUTER SCIENCE AND ENGINEERING

at

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Guindy, Chennai - 600036

JANUARY 2012

TABLE OF CONTENTS

[c1] give problem definition etc

[c2] *minor*: (i)make names in bib file uniform style w.r.t. firstname/initials (ii) have back refs - i.e. pages that cite the item.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 3

Tree Path Labeling of Path Hypergraphs - the New Results

[c1] This chapter documents all the new results obtained by us in the area of tree path labeling of path hypergraphs which is the parent problem addressed in this thesis.

[c1] UPDATE TO FINAL VERSION. BELOW IS OLDER VERSION WITH INCOMPLETE SEC 4.

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 the characterization of a feasible path labeling for any path labeling from any target trees. Section 3.4 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 3.4.1. Section 3.4.2 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 3.5 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.

3.1 Introduction

In Section 1.6 we see that consecutive-ones property and its equivalent problem of interval labeling of a hypergraph is a special case of the general problem of tree path labeling of path hypergraphs. The problem of consecutive-ones property testing can be easily seen as a simple constraint satisfaction problem involving a hypergraph or a system of sets from a universe. Every column (row) of the binary matrix can be converted into a set of non-negative integers which are the indices of the rows (columns) with **1**s in that column (row). When observed in this context, if the matrix has the COP on columns (rows), a reordering of its rows (columns) will result in sets that have only consecutive integers. In other words, the sets after applying the COP row (column) permutation are intervals. In this form, one can see that this is indeed the problem of finding interval assignments to the given set system [NS09] with a single permutation of the universe (set of row or column indices for COP of columns or rows, respectively) which permutes each set to an interval. The result in [NS09] 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. c2

c2 (ICPIA) put reference to survey chapter

Naturally, intervals are paths from a tree with maximum degree two. Thus the interval assignment problem can be 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. To elaborate, the problem is defined as follows – given a hypergraph $\mathcal{F}$ from universe (hypergraph vertex set) U and a target tree T, does there exist a bijection ϕ from U to the vertices of T such that for each hyperedge when applied to its elements, ϕ gives a path on T. More formally, the problem definition is as defined 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 **true** on $(\mathcal{F}, T, \ell)$.

To characterize a feasible TPL, we consider the case were 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 ϕ when applied on any hyperedge in $\mathcal{F}$ will give the path mapped to it by the given tree path labeling ℓ. i.e., $\ell(S) = \{\phi(x) \mid x \in S\}$, for every hyperedge $S \in \mathcal{F}$.

Section 3.3 discusses Feasible Tree Path Labeling and presents an algorithmic characterization for a feasible TPL.

With respect to computing a feasible TPL, as suggested by Compute Feasible Tree Path Labeling problem, we were unable to discover an efficient algorithm for it. Hence we consider two special cases of Compute Feasible Tree Path Labeling – namely, Compute Interval Labeling and Compute k-subdivided Star Path Labeling on special target trees, intervals and k-subdivided stars, respectively. Section 3.4 discusses these problems.

COMPUTE INTERVAL LABELING

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

COMPUTE k-SUBDIVIDED STAR PATH LABELING

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

COMPUTE INTERVAL LABELING is nothing but the consecutive-ones property testing problem.

An algorithm for COMPUTE FEASIBLE TREE PATH LABELING on general trees which is less efficient than polynomial time is presented in Section 3.5. c1

c2

c1 put some more description here. refer WG11.

c2 ~~We focus on the question of generalizing the notion of an ICPIA cite-nsnrs09 to characterize feasible path assignments. We show that for a given set system $\mathcal{F}$, a tree T, and an assignment of paths from T to the sets, there is a feasible bijection between U and $V(T)$ if and only if all intersection cardinalities among any three sets (not necessarily distinct) is same as the intersection cardinality of the paths assigned to them and the input runs a filtering algorithm (described in this paper) successfully. This characterization is proved constructively and it gives a natural data structure that stores all the relevant feasible bijections between U and $V(T)$. Further, the filtering algorithm is also an efficient algorithm to test if a tree path labeling to the set system is feasible. This generalizes the result in cite-nsnrs09.~~

3.2 Preliminaries to new results

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

Hypergraph Preliminaries: The set $\mathcal{F} \subseteq (2^U \setminus \emptyset)$ is a *set system* of a universe U with $|U| = n$. The *support* of a set system $\mathcal{F}$ denoted by $supp(\mathcal{F})$ is the union of all the sets in $\mathcal{F}$; $supp(\mathcal{F}) = \bigcup_{S \in \mathcal{F}} S$. For the purposes of this paper, a set system is required to "cover" the universe; $supp(\mathcal{F}) = U$. A set system $\mathcal{F}$ can also be visualized as a *hypergraph* vertex set is $supp(\mathcal{F})$ and hyperedges are the sets in $\mathcal{F}$. The *intersection graph* $\mathbb{I}(\mathcal{F})$ of a hypergraph $\mathcal{F}$ is a graph such that its vertex set has a bijection to $\mathcal{F}$ and there exists an edge between two vertices iff their corresponding hyperedges have a non-empty intersection [Gol04].

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

[c3]A set system $\mathcal{F}$ can be alternatively represented by a *hypergraph* $\mathcal{F}_H$ whose vertex set is *supp*$(\mathcal{F})$ and hyperedges are the sets in $\mathcal{F}$. This is a known representation for interval systems in literature [BLS99, KKLV10]. We extend this definition here to path systems. Due to the equivalence of set system and hypergraph in the scope of this paper, we drop the subscript $_H$ in the notation and refer to both the structures by $\mathcal{F}$. [c3] *Text added.*

Path Hypergraph from a Tree: 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; $\mathcal{P} \subseteq \{P \mid P \subseteq V,\ T[P] \text{ is a path}\}$. This generalizes the fact, from the literature [BLS99, KKLV10], that intervals can be viewed as sub-paths of a path.

[c1]Due to the equivalence of set system and hypergraph in the scope of this paper, we drop the subscript $_H$ in the notation and refer to both the structures by $\mathcal{F}$. [c1] *Text added.*

If the intersection graphs of $\mathcal{F}$ and $\mathcal{P}$ (a path system) are isomorphic, $\mathbb{I}(\mathcal{F}) \cong \mathbb{I}(\mathcal{P})$, then the associated bijection $\ell : \mathcal{F} \to \mathcal{P}$ due to this isomorphism is called a *path labeling* of the hypergraph $\mathcal{F}$. Note that there are two kinds of isomorphisms here. One is the isomorphism of intersection graphs on $\mathcal{F}$ and $\mathcal{P}$, i.e. $\mathbb{I}(\mathcal{F})$ and $\mathbb{I}(\mathcal{P})$ respectively. Second is the isomorphism between the hypergraphs $\mathcal{F}$ and $\mathcal{P}$.

[c2]To illustrate further, let $g : V(\mathcal{F}) \to V(\mathcal{P})$ be the above mentioned isomorphism where $V(\mathcal{F})$ and $V(\mathcal{P})$ are the vertex sets that represent the hyperedges for each hypergraph respectively, $V(\mathcal{F}) = \{v_S \mid S \in \mathcal{F}\}$ and $V(\mathcal{P}) = \{v_P \mid P \in \mathcal{P}\}$. Then the path labeling ℓ is defined as follows: $\ell(S_1) = P_1$ iff $g(v_{S_1}) = v_{P_1}$. Just to emphasize, for a path labeling ℓ of $\mathcal{F}$ with $\mathcal{P}$ as the path system, $\mathcal{F}^\ell$ is same as $\mathcal{P}$. The path system $\mathcal{P}$ may be alternatively denoted in terms of $\mathcal{F}$ and ℓ as $\mathcal{F}^\ell$. In most scenarios in this paper, what is given are the pair $(\mathcal{F}, \ell)$ and the target tree T; hence this notation will be used more often. [c2] *Text added.*

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

[c3]A graph G is a *path graph* if it is isomorphic to the intersection graph $\mathbb{I}(\mathcal{P})$ of a path system $\mathcal{P}$. This isomorphism gives a bijection $\ell' : V(G) \to \mathcal{P}$. Moreover, for the purposes of this paper, we require that in a path labeling, *supp*$(\mathcal{P}) = V(T)$. If [c3] *Text added.*

graph G is also isomorphic to $\mathbb{I}(\mathcal{F})$ for some hypergraph $\mathcal{F}$, then clearly there is a bijection $\ell : \mathcal{F} \rightarrow \mathcal{P}$ such that $\ell(S) = \ell'(v_S)$ where v_S is the vertex corresponding to set S in $\mathbb{I}(\mathcal{F})$ for any $S \in \mathcal{F}$. This bijection ℓ is called the *path labeling* of the hypergraph $\mathcal{F}$ and the path system $\mathcal{P}$ may be alternatively denoted as $\mathcal{F}^\ell$.

A path labeling $(\mathcal{F}, \ell)$ is defined to be *feasible* if $\mathcal{F} \cong \mathcal{F}^\ell$ and this hypergraph isomorphism $\phi : supp(\mathcal{F}) \rightarrow supp(\mathcal{F}^\ell)$ induces a path labeling $\ell_\phi : \mathcal{F} \rightarrow \mathcal{F}^\ell$ such that $\ell_\phi = \ell$. [c1]In this work, we are given as input $\mathcal{F}$ and a tree T, and the question is whether there is a path labeling ℓ to a set of paths in T. We refer to such a solution path system by $\mathcal{F}^\ell$. A path labeling $(\mathcal{F}, \ell)$ is defined to be *feasible* if there is a hypergraph isomorphism $\phi : supp(\mathcal{F}) \rightarrow supp(\mathcal{F}^\ell) = V(T)$ induces a path labeling $\ell_\phi : \mathcal{F} \rightarrow \mathcal{F}^\ell$ such that $\ell_\phi = \ell$.

[c1] *Text added.*

Overlap Graphs and Marginal Hyperedges: An *overlap graph* $\mathbb{O}(\mathcal{F})$ of a hypergraph $\mathcal{F}$ is a graph such that its vertex set has a bijection to $\mathcal{F}$ and there exists an edge between two of its vertices iff their corresponding hyperedges overlap. Two hyperedges S and S' are said to *overlap*, denoted by $S \between S'$, if they have a non-empty intersection and neither is contained in the other; $S \between S'$ iff $S \cap S' \neq \emptyset, S \nsubseteq S', S' \nsubseteq S$. Thus $\mathbb{O}(\mathcal{F})$ is a spanning subgraph of $\mathbb{I}(\mathcal{F})$ and not necessarily connected. Each connected component of $\mathbb{O}(\mathcal{F})$ is called an *overlap component*.

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

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

— WG11 begin —

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

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

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

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

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

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

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

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

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

— WG11 end —

3.3 Characterization of Feasible Tree Path Labeling

In this section we give an algorithmic characterization of a feasibility of tree path labeling. Consider a path labeling $(\mathcal{F}, \ell)$ on the given tree T. We call $(\mathcal{F}, \ell)$ an *Intersection Cardinality Preserving Path Labeling (ICPPL)* if it has the following properties.

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

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

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

The following lemma is useful in subsequent arguments.

Lemma 3.3.1. *If ℓ is an ICPPL, and $S_1, S_2, S_3 \in \mathcal{F}$, then $|S_1 \cap (S_2 \setminus S_3)| = |\ell(S_1) \cap (\ell(S_2) \setminus \ell(S_3))|$.*

Proof. Let $P_i = \ell(S_i)$, for all $1 \leq i \leq 3$. $|S_1 \cap (S_2 \setminus S_3)| = |(S_1 \cap S_2) \setminus S_3| = |S_1 \cap S_2| - |S_1 \cap S_2 \cap S_3|$. Due to properties (ii) and (iii) of ICPPL, $|S_1 \cap S_2| - |S_1 \cap S_2 \cap S_3| = |P_1 \cap P_2| - |P_1 \cap P_2 \cap P_3| = |(P_1 \cap P_2) \setminus P_3| = |P_1 \cap (P_2 \setminus P_3)|$. Thus lemma is proven. □

In the remaining part of this section we show that $(\mathcal{F}, \ell)$ is feasible if and only if it is an ICPPL and Algorithm 3 returns a non-empty function. Algorithm 3 recursively does two levels of filtering of $(\mathcal{F}, \ell)$ to make it simpler while retaining the set of isomorphisms, if any, between $\mathcal{F}$ and $\mathcal{F}^{\ell}$. First, we present Algorithm 1 or `filter_1`, and prove its correctness. This algorithm refines the path labeling by processing pairs of paths in $\mathcal{F}^{\ell}$ that share a leaf until no two paths in the new path labeling share any leaf.

Algorithm 1 Refine ICPPL `filter_1` $(\mathcal{F}, \ell, T)$

```
 1: $\mathcal{F}_0 \leftarrow \mathcal{F}$, $\ell_0(S) \leftarrow \ell(S)$ for all $S \in \mathcal{F}_0$
 2: $j \leftarrow 1$
 3: while there is $S_1, S_2 \in \mathcal{F}_{j-1}$ such that $\ell_{j-1}(S_1)$ and $\ell_{j-1}(S_2)$ have a common
    leaf in $T$ do
 4:     $\mathcal{F}_j \leftarrow (\mathcal{F}_{j-1} \setminus \{S_1, S_2\}) \cup \{S_1 \cap S_2, S_1 \setminus S_2, S_2 \setminus S_1\}$
        | Remove $S_1$, $S_2$ and add the
        | ``filtered'' sets
 5:     for every $S \in \mathcal{F}_{j-1}$ s.t. $S \neq S_1$ and $S \neq S_2$ do $\ell_j(S) \leftarrow \ell_{j-1}(S)$ end for
                                              | Carry forward the path
 6:     $\ell_j(S_1 \cap S_2) \leftarrow \ell_{j-1}(S_1) \cap \ell_{j-1}(S_2)$   | 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:     if $(\mathcal{F}_j, \ell_j)$ does not satisfy (Property iii) of ICPPL then
10:         exit
11:     end if
12:     $j \leftarrow j + 1$
13: end while
14: $\mathcal{F}' \leftarrow \mathcal{F}_j$, $\ell' \leftarrow \ell_j$
15: return $(\mathcal{F}', \ell')$
```

Lemma 3.3.2. *In Algorithm 1, if input $(\mathcal{F}, \ell)$ is a feasible path assignment then at the end of jth iteration of the* **while** *loop, $j \geq 0$, $(\mathcal{F}_j, \ell_j)$ is a feasible path assignment.*

Proof. We will prove this by mathematical induction on the number of iterations. The base case $(\mathcal{F}_0, \ell_0)$ is feasible since it is the input itself which is given to be feasible. Assume the lemma is true till $j-1$th iteration. i.e. every hypergraph isomorphism $\phi : supp\,(\mathcal{F}_{j-1}) \rightarrow V\,(T)$ that defines $(\mathcal{F}, \ell)$'s feasibility, is such that the induced path labeling on $\mathcal{F}_{j-1}$, $\ell_{\phi[\mathcal{F}_{j-1}]}$ is equal to ℓ_{j-1}. We will prove that ϕ is also the bijection that makes $(\mathcal{F}_j, \ell_j)$ feasible. Note that $supp(\mathcal{F}_{j-1}) = supp(\mathcal{F}_j)$ since the new sets in $\mathcal{F}_j$ are created from basic set operations to the sets in $\mathcal{F}_{j-1}$. For the same reason and ϕ being a bijection, it is clear that when applying the ϕ induced path labeling on $\mathcal{F}_j$, $\ell_{\phi[\mathcal{F}_j]}(S_1 \setminus S_2) = \ell_{\phi[\mathcal{F}_{j-1}]}(S_1) \setminus \ell_{\phi[\mathcal{F}_{j-1}]}(S_2)$. Now observe that $\ell_j(S_1 \setminus S_2) = \ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2) = \ell_{\phi[\mathcal{F}_{j-1}]}(S_1) \setminus \ell_{\phi[\mathcal{F}_{j-1}]}(S_2)$. Thus the induced path labeling $\ell_{\phi[\mathcal{F}_j]} = \ell_j$. Therefore lemma is proven. □

— WG11 begin —

Lemma 3.3.3. *If $\mathcal{A}$ is an ICPPA, and $(S_1, P_1), (S_2, P_2), (S_3, P_3) \in \mathcal{A}$, then $|S_1 \cap (S_2 \setminus S_3)| = |P_1 \cap (P_2 \setminus P_3)|$.*

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

Lemma 3.3.4. *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 $Q \cap (Q_1 \setminus Q_2) \neq \emptyset$, 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 □

Lemma 3.3.5. *In Algorithm 1, at the end of jth iteration, $j \geq 0$, of the while loop of Algorithm 1, the following invariants are maintained.*

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

Proof. Proof is by induction on the number of iterations, j. In the rest of the proof, the term "new sets" will refer to the new sets added in jth iteration as defined in line 4 of Algorithm 1, i.e. the following three assignment pairs for some $(P_1, Q_1), (P_2, Q_2) \in \Pi_{j-1}$ where Q_1 and Q_2 intersect and share a leaf: $(P_1 \cap P_2, Q_1 \cap Q_2)$, or $(P_1 \setminus P_2, Q_1 \setminus Q_2)$, or $(P_2 \setminus P_1, Q_2 \setminus Q_1)$.
The base case, $\Pi_0 = \{(S_i, P_i) \mid i \in [m]\}$, is trivially true since it is the input which is an ICPPA. Assume the lemma is true till the $j - 1$ iteration. Consider jth iteration:
If (P, Q), (P', Q') and (P'', Q'') are in Π_j and Π_{j-1}, all the invariants are clearly true because they are from $j - 1$ iteration.
If (P, Q) is in Π_j and not in Π_{j-1}, then it must be one of the new sets added in Π_j. Since (P_1, Q_1) and (P_2, Q_2) are from Π_{j-1} and Q_1, Q_2 intersect and have a common leaf, it can be verified that the new sets are also paths.
By hypothesis for invariant III, invariant II also holds for (P, Q) no matter which new set in Π_j it is.
To prove invariant III, if (P, Q) and (P', Q') are not in Π_{j-1}, then they are both new sets and invariant III holds trivially (new sets are disjoint). Next consider $(P, Q), (P', Q') \in \Pi_j$ with only one of them, say (P', Q'), in Π_{j-1}. Then (P, Q) is one of the new sets added in line 4. It is easy to see that if (P, Q) is $(P_1 \cap P_2, Q_1 \cap Q_2)$, then due to invariant IV in hypothesis, invariant III becomes true in this iteration. Similarly, using lemma 3.3.3 invariant III is proven if (P, Q) is $(P_1 \setminus P_2, Q_1 \setminus Q_2)$, or $(P_2 \setminus P_1, Q_2 \setminus Q_1)$.
To prove invariant IV, consider three assignments $(P, Q), (P', Q'), (P'', Q'')$. If at least two of these pairs are in not Π_{j-1}, then they are any two of the new sets. Note that these new sets are disjoint and hence if $(P', Q'), (P'', Q'')$ are any of these sets, $|P \cap P' \cap P''| = |Q \cap Q' \cap Q''| = 0$ and invariant IV is true. Now we consider the case if at most one of $(P, Q), (P', Q'), (P'', Q'')$ is not in Π_{j-1}. If none of them are not in Π_{j-1} (i.e. all of them are in Π_{j-1}), invariant IV is clearly true.

Consider the case where exactly one of them is not in Π_{j-1}. w.l.o.g let that be (P, Q) and it could be any one of the new sets. If (P, Q) is $(P_1 \cap P_2, Q_1 \cap Q_2)$, from lemma 3.3.4 and invariant III hypothesis, invariant IV is proven. Similarly if (P, Q) is any of the other new sets, invariant IV is proven by also using lemma 3.3.3.

□

— WG11 end —

Lemma 3.3.6. *In Algorithm 1, at the end of jth iteration, $j \geq 0$, of the* **while** *loop, the following invariants are maintained.*

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

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

✠ *Invariant* I/II

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

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

$\ell_{j-1}(S_2)| = |\ell_j(S_1 \cap S_2)| = |\ell_j(R)|$. If $R = S_1 \setminus S_2$, $|R| = |S_1 \setminus S_2| = |S_1| - |S_1 \cap S_2| = |\ell_{j-1}(S_1)| - |\ell_{j-1}(S_1) \cap \ell_{j-1}(S_2)| = |\ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2)| = |\ell_j(S_1 \setminus S_2)| = |\ell_j(R)|$. Similarly if $R = S_2 \setminus S_1$.

✠ *Invariant* III

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

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

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

✠ *Invariant* IV

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

□

Lemma 3.3.7. *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. Secondly, for any iteration $j > 0$ of the* **while** *loop, the* **exit** *statement in line 10 will not execute.*

Proof. Since $(\mathcal{F}, \ell)$ is feasible, by Lemma 3.3.2 $(\mathcal{F}_j, \ell_j)$ for every iteration $j > 0$ is feasible. Also, every hypergraph isomorphism $\phi : supp(\mathcal{F}) \rightarrow V(T)$ that induces ℓ on $\mathcal{F}$ also induces ℓ_j on $\mathcal{F}_j$, i.e., $\ell_{\phi[\mathcal{F}_j]} = \ell_j$. Thus it can be seen that for all $x \in supp(\mathcal{F})$, for all $v \in V(T)$, if $(x, v) \in \phi$ then $v \in \ell_j(S)$ for all $S \in \mathcal{F}_j$ such that $x \in S$. In other words, filter 1 outputs a filtered path labeling that "preserves" hypergraph isomorphisms of the original path labeling.
Secondly, line 10 will execute iff the exit condition in line 9, i.e. failure of three way intersection preservation, becomes true in any iteration of the **while** loop.

Due to Lemma 3.3.6 Invariant IV, the exit condition does not occur if the input is a feasible ICPPL.

□

As a result of Algorithm 1 each leaf v in T is such that there is exactly one set in $\mathcal{F}$ with v as a vertex in the path assigned to it. In Algorithm 2 we identify elements in *supp*($\mathcal{F}$) whose images are leaves in a hypergraph isomorphism if one exists. Let $S \in \mathcal{F}$ be such that $\ell(S)$ is a path with leaf and $v \in V(T)$ is the unique leaf incident on it. We define a new path labeling ℓ_{new} such that $\ell_{new}(\{x\}) = \{v\}$ where x an arbitrary element from $S \setminus \bigcup_{\hat{S} \neq S} \hat{S}$. In other words, x is an element present in no other set in $\mathcal{F}$ except S. This is intuitive since v is present in no other path image under ℓ other than $\ell(S)$. The element x and leaf v are then removed from the set S and path $\ell(S)$ respectively. After doing this for all leaves in T, all path images in the new path labeling ℓ_{new} except leaf labels (a path that has only a leaf is called the *leaf label* for the corresponding single element hyperedge or set) are paths from a new pruned tree $T_0 = T \setminus \{v \mid v \text{ is a leaf in } T\}$. Algorithm 2 is now presented with details.

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

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

Suppose the input ICPPL $(\mathcal{F}, \ell)$ is feasible, yet set X in Algorithm 2 is empty in some iteration of the **while** loop. This will abort our procedure of finding the hypergraph isomorphism. The following lemma shows that this will not happen.

— WG11 begin —

Lemma 3.3.8. *In Algorithm 2,for all $j \geq 0$, at the end of the jth iteration the four invariants given in lemma 3.3.6 are valid.*

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

We use mathematical induction on the number of iterations for this proof. The term "new sets" will refer to the sets added in Π_j in the jth iteration, i.e. $(P' \setminus \{x\}, Q' \setminus \{v\})$ and $(\{x\}, \{v\})$ for some (P', Q') in Π_{j-1} such that v is a leaf and Q' is the unique path incident on it.
For Π_0 all invariants hold because it is output from algorithm 1 which is an ICPPA. Hence base case is proved. Assume the lemma holds for Π_{j-1}. Consider Π_j and any $(P, Q) \in \Pi_j$. If (P, Q) is in Π_j and Π_{j-1} invariants I and II are true because of induction assumption. If it is only in Π_j, then it is $\{(P' \setminus \{x\}), (Q' \setminus \{v\})$ or $(\{x\}, \{v\})$ for some (P', Q') in Π_{j-1}. By definition, x is an element in P' (as defined in the algorithm) and v is a leaf in Q'. If (P, Q) is $\{(P' \setminus \{x\}), (Q' \setminus \{v\}), Q$ is a path since only a leaf is removed from path Q'. We know $|P'| = |Q'|$, therefore $|P' \setminus \{x\}| = |Q' \setminus \{v\}|$. Hence in this case invariants I and II are obvious. It is easy to see these invariants hold if (P, Q) is $(\{x\}, \{v\})$.

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

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

$P_1, P_2 \notin \Pi_{j-1}$ and new sets are derived from $(P', Q'), (P'', Q'') \in \Pi_{j-1}$ with x_1, x_2 exclusively in P_1, P_2, $(\{x_1\}, \{v_1\}), (\{x_2\}, \{v_2\}) \in \Pi_j$ thus v_1, v_2 are exclusively in Q_1, Q_2 resp. it follows that $|P_1 \cap P_2 \cap P_3| = |(P' \setminus \{x_1\}) \cap (P'' \setminus \{x_2\}) \cap P_3| = |P' \cap P'' \cap P_3| = |Q' \cap Q'' \cap Q_3| = |(Q' \setminus \{v_1\} \cap Q'' \setminus \{v_2\} \cap Q_3| = |Q_1 \cap Q_2 \cap Q_3|$. Thus invariant IV is also proven. $\square$

Using algorithms 1 and 2 we prove the following theorem.

Theorem 3.3.9. *If $\mathcal{A}$ is an ICPPA, then there exists a bijection $\sigma : U \to V(T)$ such that $\sigma(S_i) = P_i$ for all $i \in I$*

Proof. This is a contructive proof. First, the given ICPPA $\mathcal{A}$ and tree T are given as input to Algorithm 1. This yields a "filtered" ICPPA as the output which is input to Algorithm 2. It can be observed that the output of Algorithm 2 is a set of interval assignments to sets and one-to-one assignment of elements of U to each leaf of T. To be precise, it would be of the form $\mathcal{B}_0 = \mathcal{A}_0 \cup \mathcal{L}_0$. The leaf assignments are defined in $\mathcal{L}_0 = \{(x_i, v_i) \mid x_i \in U, v_i \in T, x_i \neq x_j, v_i \neq v_j, i \neq j, i, j \in [k]\}$ where k is the number of leaves in T. The path assignments are defined in $\mathcal{A}_0 \subseteq \{(S'_i, P'_i) \mid S'_i \subseteq U_0, P'_i$ is a path from $T_0\}$ where T_0 is the tree obtained by removing all the leaves in T and $U_0 = U \setminus \{x \mid x$ is assigned to a leaf in $\mathcal{L}_0\}$. Now we have a subproblem of finding the permutation for the path assignment $\mathcal{A}_0$ which has paths from tree T_0 and sets from universe U_0. Now we repeat the procedure and the path assignment $\mathcal{A}_0$ and tree T_0 is given as input to Algorithm 1. The output of this algorithm is given to Algorithm 2 to get a new union of path and leaf assignments $\mathcal{B}_1 = \mathcal{A}_1 \cup \mathcal{L}_1$ defined similar to $\mathcal{B}_0, \mathcal{L}_0, \mathcal{A}_0$. In general, the two algorithms are run on path assignment $\mathcal{A}_{i-1}$ with paths from tree T_{i-1} to get a new subproblem with path assignment $\mathcal{A}_i$ and tree T_i. T_i is the subtree of T_{i-1} obtained by removing all its leaves. More importantly, it gives leaf assignments $\mathcal{L}_i$ to the leaves in tree T_{i-1}. This is continued until we get a subproblem with path assignment $\mathcal{A}_{d-1}$ and tree T_{d-1} for some $d \leq n$ which is just a path. From the last lemma we know that $\mathcal{A}_{d-1}$ is an ICPPA. Another observation is that an ICPPA with all its tree paths being intervals (subpaths from a path) is nothing but an ICPIA[NS09]. Let $\mathcal{A}_{d-1}$ be equal to $\{(S''_i, P''_i) \mid S''_i \subseteq U_{d-1}, P''_i$ is a path from $T_{d-1}\}$. It is true that the paths P''_is may not be precisely an interval in the sense of consecutive integers because they are some nodes from a tree. However, it is easy to see that the nodes of T_{d-1} can be ordered from left to right and ranked to get intervals I_i for every path P''_i as follows. $I_i = \{[l, r] \mid l =$ the lowest rank of the nodes in $P''_i, r = l + |P''_i| - 1\}$. Let asssignment $\mathcal{A}_d$ be with the renamed paths. $\mathcal{A}_d = \{(S''_i, I_i) \mid (S''_i, P''_i) \in \mathcal{A}_{d-1}\}$. What has been effectively done is renaming the nodes in T_{d-1} to get a tree T_d. The

ICPIA $\mathcal{A}_d$ is now in the format that the ICPIA algorithm requires which gives us the permutation $\sigma' : U_{d-1} \rightarrow T_{d-1}$

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

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

To summarize, run algorithm 1 and 2 on T. After the leaves have been assigned to specific elements from U, remove all leaves from T to get new tree T_0. The leaf assignments are in $\mathcal{L}_0$. Since only leaves were removed T_0 is indeed a tree. Repeat the algorithms on T_0 to get leaf assignments $\mathcal{L}_1$. Remove the leaves in T_0 to get T_1 and so on until the pruned tree T_d is a single path. Now run ICPIA algorithm on T_d to get permutation σ'. The relation $\mathcal{L}_0 \cup \mathcal{L}_1 \cup .. \cup \mathcal{L}_d \cup \sigma'$ gives the bijection required in the original problem. □

— WG11 end —

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* **exit** *statement in line 8 does not execute.*

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

Lemma 3.3.11. *In Algorithm 2, for all $j > 0$, at the end of the jth iteration of the* **while** *loop the four invariants given in Lemma 3.3.6 hold.*

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

We have seen two filtering algorithms above, namely, Algorithm 1 `filter_1` and Algorithm 2 `filter_2` which when executed serially repectively result in a new ICPPL on the same universe U and tree T. We also proved that if the input is indeed feasible, these algorithms do indeed output the filtered ICPPL. Now we present the algorithmic characterization of a feasible tree path labeling by way of Algorithm 3.

Algorithm 3 computes a hypergraph isomorphism ϕ recursively using Algorithm 1 and Algorithm 2 and pruning the leaves of the input tree. In brief, it is done as follows. Algorithm 2 gives us the leaf labels in $\mathcal{F}_2$, i.e., the elements in *supp*$(\mathcal{F})$ that map to leaves in T, where $(\mathcal{F}_2, \ell_2)$ is the output of Algorithm 2. All leaves in T are then pruned away. The leaf labels are removed from the path labeling ℓ_2 and the corresponding elements are removed from the corresponding sets in $\mathcal{F}_2$. This tree pruning algorithm is recursively called on the altered hypergraph $\mathcal{F}'$, path label ℓ' and tree T'. The recursive call returns the bijection ϕ'' for the rest of the elements in *supp*$(\mathcal{F})$ which along with the leaf labels ϕ' gives us the hypergraph isomorphism ϕ. The following lemma formalizes the characeterization of feasible path labeling.

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

```
1: if $T$ is empty then
2:     return $\emptyset$
3: end if
4: $L \leftarrow \{v \mid v \text{ is a leaf in } T\}$
5: $(\mathcal{F}_1, \ell_1) \leftarrow$ filter_1$(\mathcal{F}, \ell, T)$
6: $(\mathcal{F}_2, \ell_2) \leftarrow$ filter_2$(\mathcal{F}_1, \ell_1, T)$
7: $(\mathcal{F}', \ell') \leftarrow (\mathcal{F}_2, \ell_2)$
8: $\phi' \leftarrow \emptyset$
9: for every $v \in L$ do
10:     $\phi'(x) \leftarrow v$ where $x \in \ell_2^{-1}(\{v\})$    ▷ Copy the leaf labels to a one to one function $\phi' : supp(\mathcal{F}) \rightarrow L$
11:     Remove $\{x\}$ and $\{v\}$ from $\mathcal{F}'$, $\ell'$ appropriately
12: end for
13: $T' \leftarrow T \setminus L$
14: $\phi'' \leftarrow$ get-hypergraph-isomorphism$(\mathcal{F}', \ell', T')$
15: $\phi \leftarrow \phi'' \cup \phi'$
16: return $\phi$
```

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

Proof. It is clear that in the end of every recursive call to Algorithm 3, the function ϕ' is one to one involving all the leaves in the tree passed as input to that recursive call. Moreover, by Lemma 3.3.7 and Lemma 3.3.10 it is consistent with the tree path labeling ℓ passed. The tree pruning is done by only removing leaves in each call to the function and is done till the tree becomes empty. Thus the returned function ϕ : *supp*$(\mathcal{F}) \rightarrow V(T)$ is a union of mutually exclusive one to one functions exhausting all vertices of the tree. In other words, it is a bijection from *supp*$(\mathcal{F})$ to $V(T)$ inducing the given path labeling ℓ and thus a hypergraph isomorphism. □

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

Proof. From Lemma 3.3.12, we know that if $(\mathcal{F}, \ell)$ is an ICPPL and Algorithm 3 with $(\mathcal{F}, \ell, T)$ as input returns a non-empty function, $(\mathcal{F}, \ell)$ is feasible. Now consider the case where $(\mathcal{F}, \ell)$ is feasible. i.e. there exists a hypergraph isomorphism ϕ such that $\ell_\phi = \ell$. Lemma 3.3.7 and Lemma 3.3.10 show us that filter 1 and filter 2 do not exit if input is feasible. Thus Algorithm 3 returns a non-empty function. □

3.4 Computing feasible TPL with special target trees [c1]

[c1] give problem definition etc

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[c1]. In other words, the problem is to find out if a feasible tree path labeling exists from a given target tree for a given hypergraph. In this section we will see two special cases of this problem where the target tree is from a particular family of trees. The first one, where the tree is a path, is a shown to be equivalent to the well studied problem of consecutive-ones in Section 3.4.1. The second one, where the tree is a k-subdivided tree[c1], has been solved using a polynomial time algorithm. The latter problem enforces some conditions on the hypergraph too which will be seen in Section 3.4.2.

[c1]A larger problem would be to find if a given hypergraph is a path graph on any tree. This problem is not addressed in this thesis.

[c1]See Section 3.2 for the formal definition.

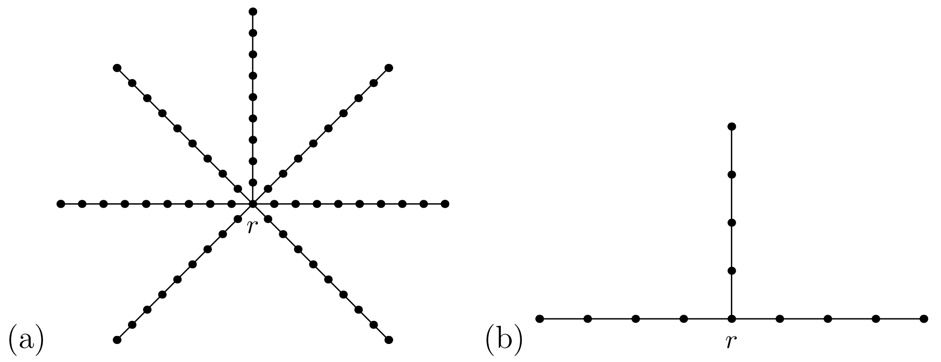

Figure 3.1: (a) 8-subdivided star with 7 rays (b) 3-subdivided star with 3 rays

3.4.1 Target tree is a Path

Consider a special case of ICPPL with the following properties when the tree T is a path. Hence, all path labels are can be viewed as intervals assigned to the sets in $\mathcal{F}$. It is shown, in [NS09], that the filtering algorithms outlined above need only preserve pairwise intersection cardinalities, and higher level intersection cardinalities are preserved by the Helly Property of intervals. Consequently, the filter algorithms do not need to ever evaluate the additional check to **exit**. This structure and its algorithm is used in the next section for finding tree path labeling from a k-subdivided star due to this graph's close relationship with intervals.

3.4.2 Target tree is a k-subdivided Star

In this section we consider the problem of assigning paths from a k-subdivided star T to a given set system $\mathcal{F}$. We consider $\mathcal{F}$ for which the overlap graph $\mathbb{O}(\mathcal{F})$ is connected. The overlap graph is well-known from the work of [KKLV10, NS09, Hsu02]. We use the notation in [KKLV10]. Recall from Section 3.2 that hyperedges S and S' are said to overlap, denoted by $S \between S'$, if S and S' have a non-empty intersection but neither of them is contained in the other. The overlap graph $\mathbb{O}(\mathcal{F})$ is a graph in which the vertices correspond to the sets in $\mathcal{F}$, and the vertices corresponding to the hyperedges S and S' are adjacent if and only if they overlap. Note that the intersection graph of $\mathcal{F}$, $\mathbb{I}(\mathcal{F})$ is different from $\mathbb{O}(\mathcal{F})$ and $\mathbb{O}(\mathcal{F}) \subseteq \mathbb{I}(\mathcal{F})$. A connected component of $\mathbb{O}(\mathcal{F})$ is called an overlap component of $\mathcal{F}$. An interesting property of the overlap components is that any two distinct overlap components, say $\mathcal{O}_1$ and $\mathcal{O}_2$, are such that any two sets $S_1 \in \mathcal{O}_1$ and $S_2 \in \mathcal{O}_2$ are disjoint, or, w.l.o.g, all the sets in $\mathcal{O}_1$ are contained within one set in $\mathcal{O}_2$. This containment relation naturally determines a decomposition of the overlap components into rooted containment trees. We consider the case when there is only one rooted containment tree, and we first present our algorithm when $\mathbb{O}(\mathcal{F})$ is connected. It is easy to see that once the path labeling to the overlap component in the root of the containment tree is achieved, the path labeling to the other overlap components in the rooted containment tree is essentially finding a path labeling when the target tree is a path: each target path is a path that is allocated to sets in the root overlap component. Therefore, for the rest of this section, $\mathbb{O}(\mathcal{F})$ is a connected graph. We also assume that all hyperedges are of cardinality at most $k+2$.

Recall from Section 3.2 that a k-subdivided star is a star with each edge subdivided k times. Therefore, a k-subdivided star has a central vertex which we call the *root*, and each root to leaf path is called a *ray*. First, we observe that by removing the root r from T, we get a collection of p vertex disjoint paths of length $k+1$, p being the number of leaves in T. We denote the rays by $R_1, \ldots, R_p$, and the number of vertices in R_i, $i \in [p]$ is $k+2$. Let $\left\langle v_{i1}, \ldots, v_{i(k+2)} = r \right\rangle$ 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 ith iteration. Note that $\mathcal{O}_1 = \mathcal{O}$. In the ith 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.

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 $R_1, \ldots, R_p$, and the number of vertices in R_i, $i \in [p]$ is $k+2$. Let $\langle v_{i1}, \ldots, v_{i(k+2)} = r \rangle$ denote the sequence of vertices in R_i, where v_{i1} is the leaf. Note that r is a common vertex to all R_i.

3.4.3 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 classifed into the following disjoint sets.

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

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

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

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

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

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

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

II. **Iteration i:** Let X be a hyperedge from $\mathcal{O}_i$ such that there exists $Y \in \mathcal{L}_1^i \cup \mathcal{L}_2^i$ and $X \between Y$. Further let $Z \in \mathcal{L}_1^i \cup L_2^i$ such that $Z \between Y$. If $X \in \mathcal{T}_2^i$, and if there are multiple Y candidates then any Y is selected. On the other hand, if $X \in \mathcal{T}_1^i$, then X has a partial path assignment, $\ell'(X)$ from a previous iteration, say from ray R_j. Then, Y is chosen such that $X \cap Y$ has a non-empty intersection with a ray different from R_j. The key things that are done in assigning a path to X are as follows. The *end* of path $\ell(Y)$ where $\ell(X)$ would overlap is found, and then based on this the existence of a feasible assignment is decided. It is important to note that since $X \between Y$, $\ell(X) \between \ell(Y)$ in any feasible assignment. Therefore, the notion of the *end* at which $\ell(X)$ and $\ell(Y)$ overlap is unambiguous, since for any path, there are two end points.

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

(b) *Any path of length $s(X)$ at the appropriate end contains r:* If $X \in \mathcal{T}_1^i$ then after finding the appropriate end as in step **??** 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 **??**, $\ell(X)$ 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 involes 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 3.4.1. *If $X \in \mathcal{F}$ is super-marginal and ℓ is a feasible tree path labeling to tree T, then $\ell(X)$ will contain a leaf in T.*

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

Lemma 3.4.2. *If $\mathcal{O}$ does not have any super-marginal edges, then in any feasible path labeling ℓ of $\mathcal{O}$ with paths from T is such that, for any hyperedge X for which $\ell(X)$ contains a leaf, $|X| \geq k+3$.*

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

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

Lemma 3.4.3. *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 3.4.4. *Given $\mathcal{O}$ and a k-subdivided star T, the above algorithm decides correctly if there is a feasible path labeling ℓ.*

Proof. Outline. If the algorithm outputs a path labeling ℓ, then it is clear that it is an ICPPL. The reason is that the algorithm checks that three-way intersection cardinalities are preserved in each iteration which ensures ICPPL Property iii. Moreover, it is clear that $\ell(X)$ for any $X \in \mathcal{O}$ is computed by maintaining ICPPL Property i and ICPPL Property ii. For such a labeling ℓ, the proof that it is feasible is by induction on k. What needs to be shown is that Algorithm 3 successfully runs on input $(\mathcal{O}, \ell)$. In base case $k = 0$, T is a star. Also every set is at most size 2 $(k+2)$ size and thus overlaps are at most 1. If two paths share a leaf in `filter_1` 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_2` 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_1` and one iteration of a modified `filter_2` *all* leaves are assigned pre-images. Removing the leaves from T and the pre-images from support of $\mathcal{O}$, results in an ICPPL to a $(k-1)$-subdivided star. Now we apply the induction hypothesis, and we get an isomorphism between the hypergraphs $\mathcal{O}$ and $\mathcal{O}^\ell$.

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

3.5 TPL with no restrictions

`abstract begin` 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. [c1] `abstract end`

[c1] [TRUE?]

In Section **??** we present the necessary preliminaries, in Section **??** we present our characterization of feasible tree path assignments, and in Section 3.5.1 we present the characterizing subproblems for finding a bijection between U and $V(T)$ such that sets map to tree paths. Finally, in Section 3.6 we conclude by showing that Tree Path Assignment is GI-Complete, and also observe that Consecutive Ones Testing is in Logspace.

[c2] We refer to this as the Tree Path Assignment problem for an input $(\mathcal{F}, T)$

[c2] Add this "pair" as a definition in ch:prelims

pair.

[c3] In the second part of this paper, we decompose our search for a bijection between U and $V(T)$ into subproblems. Each subproblem is on a set system in which for each set, there is another set in the set system with which the intersection is *strict*- there is a non-empty intersection, but neither is contained in the other. This is in the spirit of results in [Hsu02, NS09] where to test for COP in a given matrix, the COP problem is solved on an equivalent set of prime matrices. [c4]Our decomposition localizes the challenge of path graph isomorphism to two problems.

[c3] ~~It is an interesting fact that for a matrix with the COP, the intersection graph of the corresponding set system is an interval graph. A similar connection to a subclass of chordal graphs, and this subclass contains interval graphs, exists for the generalization of COP. In this case, the intersection graph of the corresponding set system must be a path graph. Chordal graphs are of great significance, extensively studied, and have several applications. One of the well known and interesting properties of a chordal graphs is its connection with intersection graphs citemcg04. For every chordal graph, there exists a tree and a family of subtrees of this tree such that the intersection graph of this family is isomorphic to the chordal graph citeplr70,gav78,bp9 Certain format of these trees are called clique trees citeapy92 of the graph which is a compact representation of the chordal graph. There has also been work done on the generalization of clique trees to clique hypergraphs citekm02. If the chordal graph can be represented as the intersection graph of paths in a tree, then the graph is called path graph citemcg04 or path chordal graph. Therefore, it is clear that if there is a bijection from U to $V(T)$ such that the sets map to paths, then the intersection graph of the set system is indeed a path chordal graph. This is, however, only a necessary condition and can be checked efficiently, as path graph recognition is polynomial solvable cite-gav78,aas93. Indeed, it is possible to construct a set system and tree, such that the intersection graph is a path chordal graph, but there is no bijection~~

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

3.5.1 Finding an assignment of tree paths to a set system

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

A set system can be concisely represented by a binary matrix where the row indices denote the universe of the set system and the column indices denote each of the sets. Let the binary matrix be M with order $n \times m$, the set system be $\mathcal{F} = \{S_i \mid i \in [m]\}$, universe of set system $U = \{x_1, \ldots, x_n\}$. If M represents $\mathcal{F}$, $|U| = n, |\mathcal{F}| = m$. Thus (i,j)th element of M, $M_{ij} = 1$ iff $x_i \in S_j$. If $\mathcal{F}$ has a feasible tree path assignment (ICPPA) $\mathcal{A} = \{(S_i, P_i) \mid i \in [m]\}$, then we say its corresponding matrix M has an ICPPA. Conversly we say that a matrix M has an ICPPA if there exists an ICPPA $\mathcal{A}$ as defined above.
We now define the strict intersection graph or overlap graph of $\mathcal{F}$. This graph occurs at many places in the literature, see for example [KKLV10, Hsu02, NS09]. The vertices of the graph correspond to the sets in $\mathcal{F}$. An edge is present between vertices of two sets iff the corresponding sets have a nonempty intersection and none is contained in the other. Formally, intersection graph is $G_f = (V_f, E_f)$ such that $V_f = \{v_i \mid S_i \in \mathcal{F}\}$ and $E_f = \{(v_i, v_j) \mid S_i \cap S_j \neq \emptyset \text{ and } S_i \nsubseteq S_j, S_j \nsubseteq S_i\}$. We use this graph to decompose M as described in [Hsu02, NS09]. A prime submatrix of M is defined as the matrix formed by a set of columns of M which

correspond to a connected component of the graph G_f. Let us denote the prime sub-matrices by $M_1, \ldots, M_p$ each corresponding to one of the p components of G_f. Clearly, two distinct matrices have a distinct set of columns. Let $col(M_i)$ be the set of columns in the sub-matrix M_i. The support of a prime sub-matrix M_i is defined as $supp(M_i) = \bigcup_{j \in col(M_i)} S_j$. Note that for each i, $supp(M_i) \subseteq U$. For a set of prime sub-matrices X we define $supp(X) = \bigcup_{M \in X} supp(M)$.

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

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

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

Lemma 3.5.1. *Let $(M_i, M_j) \in \preccurlyeq$. Then there is a set $S' \in M_j$ such that for each $S \in M_i$, $S \subseteq S'$.*

Lemma 3.5.2. *For each pair of prime sub-matrices, either $(M_i, M_j) \notin \preccurlyeq$ or $(M_j, M_i) \notin \preccurlyeq$.*

Lemma 3.5.3. *If $(M_i, M_j) \in \preccurlyeq$ and $(M_j, M_k) \in \preccurlyeq$, then $(M_i, M_k) \in \preccurlyeq$.*

Lemma 3.5.4. *If $(M_i, M_j) \in \preccurlyeq$ and $(M_i, M_k) \in \preccurlyeq$, then either $(M_j, M_k) \in \preccurlyeq$ or $(M_k, M_j) \in \preccurlyeq$.*

Theorem 3.5.5. *$\preccurlyeq$ is a partial order on the set of prime sub-matrices of M. Further, it uniquely partitions the prime sub-matrices of M such that on each set in the partition $\preccurlyeq$ induces a total order.*

For the purposes of this paper, we refine the total order mentioned in Theorem 3.5.5. We do this by identifying an in-tree rooted at each maximal upper bound under $\preccurlyeq$. Each of these in-trees will be on disjoint vertex sets, which in this case would be disjoint sets of prime-submatrices. The in-trees are specified by selecting the appropriate edges from the Hasse diagram associated with $\preccurlyeq$. Let $\mathcal{I}$ be the following set:

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

Theorem 3.5.6. *Consider the directed graph X whose vertices correspond to the prime sub-matrices, and the edges are given by $\mathcal{I}$. Then, X is a vertex disjoint collection of in-trees and the root of each in-tree is a maximal upper bound in $\preccurlyeq$.*

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

Let the partition of X given by Theorem 3.5.6 be $\{X_1, \ldots, X_r\}$. Further, each in-tree itself can be layered based on the distance from the root. The root is considered to be at level zero. For $j \geq 0$, Let $X_{i,j}$ denote the set of prime matrices in level j of in-tree X_i.

Lemma 3.5.7. *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 $\{X_1, \ldots, X_r\}$ be the partition of X into in-trees as defined above. Then, matrix M has an ICPPA in tree T iff T can be partitioned into vertex disjoint subtrees $\{T_1, T_2, \ldots T_r\}$ such that, for each $1 \leq i \leq r$, the set of prime sub-matrices corresponding to vertices in X_i has an ICPPA in T_i.*

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

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

We call this sub-tree T_i. All other prime-submatrices in X_i are assigned paths in T_i since $\mathcal{A}$ is an ICPPA, and the support of other prime sub-matrices in X_i are contained in the support of the matrix in $X_{i,0}$. Secondly, for each $1 \leq i \neq j \leq r$, $supp(X_i) \cap supp(X_j) = \phi$, and since $\mathcal{A}$ is an ICPPA, it follows that T_i and T_j are vertex disjoint. Finally, since $|U| = |V(T)|$, it follows that $T_1, \ldots, T_r$ is a partition of T into vertex disjoint sub-trees such that for each $1 \leq i \leq r$, the set of matrices corresponding to nodes in X_i has an ICPPA in T_i. Hence the lemma. □

The essence of the following lemma is that an ICPPA only needs to be assigned to the prime sub-matrix corresponding to the root of each in-tree, and all the other prime sub-matrices only need to have an Intersection Cardinality Preserving Interval Assignments (ICPIA). Recall, an ICPIA is an assignment of intervals to sets such that the cardinality of an assigned interval is same as the cardinality of the interval, and the cardinality of intersection of any two sets is same as the cardinality of the intersection of the corresponding intervals. It is shown in [NS09] that the existence of an ICPIA is a necessary and sufficient condition for a matrix to have COP. We present the pseudo-code to test if M has an ICPPA in T.

Lemma 3.5.8. *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 $\{X_1, \ldots, X_r\}$ be the partition of X into in-trees as defined earlier in this section. Let T be the given tree and let $\{T_1, \ldots, T_r\}$ be a given partition of T into vertex disjoint sub-trees. Then, for each $1 \leq i \leq r$, the set of matrices corresponding to vertices of X_i has an ICPPA in T_i if and only if the matrix in $X_{i,0}$ has an ICPPA in T_i and all other matrices in X_i have an* ***ICPIA****.*

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

Lemma 3.5.7 and Lemma 3.5.8 point out two algorithmic challenges in finding an ICPPA for a given set system $\mathcal{F}$ in a tree T. Given $\mathcal{F}$, finding X and its partition $\{X_1, \ldots, X_r\}$ into in-trees can be done in polynomial time. On the other hand, as per lemma 3.5.7 we need to parition T into vertex disjoint sub-trees

$\{T_1, \ldots, T_r\}$ such that for each i, the set of matrices corresponding to nodes in X_i have an ICPPA in T_i. This seems to be a challenging step, and it must be remarked that this step is easy when T itself is a path, as each individual T_i would be sub-paths. The second algorithmic challenge is identified by lemma 3.5.8 which is to assign an ICPPA from a given tree to the matrix associated with the root node of X_i.

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

```
Identify the prime sub-matrices. This is done by constructing the strict overlap graph and identify connected components. Each connected component yields a prime sub-matrix.
Construct the partial order $\preccurlyeq$ on the set of prime sub-matrices.
Construct the partition $X_1, \ldots, X_r$ of the prime sub-matrices induced by $\preccurlyeq$
For each $1 \leq i \leq r$, Check if all matrices except those in $X_{i,0}$ has an ICPIA.
If a matrix does not have ICPIA exit with a negative answer. To check for the existence of ICPIA, use the result in [NS09].
Find a partition of $T_1, \ldots, T_r$ such that matrices in $X_{i,0}$ has an ICPPA in $T_i$. If not such partition exists, exit with negative answer.
```

3.6 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.

3.6.1 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 iff the matrix has COP. Indeed, this is an algorithmic characterization of the consecutive ones property and the absence of the PQ-tree does not yield any witness to the reason for failure. A closely related data structure is the generalized

PQ tree in [McC04]. In generalized PQ tree the P and Q nodes are called prime and linear nodes. Aside from that, it has a third type of node called degenerate nodes which is present only if the set system does not have COP [McC04]. Using the idea of generalized PQ tree, [McC04] proves that checking for bipartiteness in the certain incomparability graph is sufficient to check for COP. [McC04] invented a certificate to confirm when a binary matrix does not have COP. [McC04] describes a graph called incompatibility graph of a set system $\mathcal{F}$ which has vertices $(a, b), a \neq b$ for every $a, b \in U$, U being the universe of the set system. There are edges $((a, b), (b, c))$ and $((b, a), (c, b))$ if there is a set $S \in \mathcal{F}$ such that $a, c \in S$ and $b \notin S$. In other words the vertices of an edge in this graph represents two orderings that cannot occur in a consecutive ones ordering of $\mathcal{F}$.

Theorem 3.6.1. *[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 sub-matrix that does not have an ICPIA. Further, the above theorem can be used to check if a given matrix has COP in logspace by checking if its incompatibility graph is bipartite. [Rei84] showed that checking for bipartiteness can be done in logspace. Thus we conclude that consecutive ones testing can be done in logspace.

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

Theorem 3.6.2. *[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 $\{1, 2, \ldots n\}$. The edge set of $\mathcal{H}$ is isomorphic to $\mathcal{F}$. Thus every edge in $\mathcal{H}$ represents a set in the given set system $\mathcal{F}$. Let this mapping be $\pi : E(\mathcal{H}) \to \mathcal{F}$. It is easy to see that if M has COP, then $\mathcal{H}$ is an interval hypergraph. From theorem 3.6.2, it is clear that the interval labeling $l_{\mathcal{H}} : V(\mathcal{H}) \to [n]$ 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 3.6.3. *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.

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[JJLM97] Michael Jünger, Michael Junger, Sebastian Leipert, and Petra Mutzel. Pitfalls of using PQ-trees in automatic graph drawing, 1997.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[YC95] Yu and Chen. Efficient parallel algorithms for doubly convex-bipartite graphs. *TCS: Theoretical Computer Science*, 147:249–265, 1995.