# 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**


**DECEMBER 2011**

# THESIS CERTIFICATE

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

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

Chennai
30 December 2011

# ACKNOWLEDGEMENTS

c1

c1 WIP

# ABSTRACT

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

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

c1

c1 EXPAND. Abstract must be a brief about what results we have and how it fits in the body of research. – Area: Broad to Specialized. i.e. Combinatorial algorithms -> Matrix reorganization -> general data reorganization (interval assignment) -> path assignment – Class of problems: say, data reorganization. – Nature of results: Is a generalization. We have a Polynomial algorithm for a subset of the generalization.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABBREVIATIONS

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

# NOTATION

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

# CHAPTER 1

# Introduction

c1

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

c1 The main themes of the thesis that must not be left out:
connection of COP to ||set systems|| and thus to general data bases. cite:nsnrs09
||graph isomorphism|| by canonization cite:kklv10
||certificate for a problem|| cite:mcc04
Also see purple book notes.

c2 Section 1.3 gives a brief survey of COP and optimization problems related to it followed by motivation for the thesis in Section 1.4. Section 1.5 presents a summary of our results on the extension of COP namely, the tree path labeling problem. c3

c2 Have a para on Organization: Outline of document

c3 *TH*: update for thesis

## 1.1 Illustration of the problem

A group of students, **Pa***tricia*, **Pi***gpen*, **Sn***oopy*, **Wo***odstock*, **Vi***olet*, **Li***nus*, **Ch***arlie*, **Sa***lly*, **Fr***anklin*, **Sc***hröeder* and **Lu***cy* enroll at the *Wallace Studies Institute* for a liberal arts programme. As part of their semester thesis, they pick a body of work to study and form the namesake study groups, *"𝔹rief Interviews with Hideous Men"*, *"The String 𝕋heory"*, *"[𝕎]Rhetoric and the Math Melodrama"* and *"𝔽ate, Time, and Language: An Essay on Free Will"*[c4]. A student will be in at least one study group and may be in more than one. For instance, as will be seen later, **Fr***anklin* studies both *"𝔹rief Interviews with Hideous Men"* and *"𝔽ate, Time, and Language: An Essay on Free Will"* while **Wo***odstock* studies only *"[𝕎]Rhetoric and the Math Melodrama"*.

c4 *TH*: put bib entries for these works!

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

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

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

The campus has a residential area *Infinite Loop* that has $n$ single occupancy apartments reserved for the study groups' accommodation. All these apartments are located such that the streets connecting them do *not* form loops. Fig 1.2 [c5] shows the street map for *Infinite Loop*. It may be noted that as a graph, it classifies as a tree.

[c5] *TH*: fig:streetmap

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

*Apartment allocation* ($\phi$)

| | |
|---|---|
| 1 | **Sa** |
| 2 | **Pi** |
| 3 | **Fr** |
| 4 | **Wo** |
| 5 | **Ch** |
| 6 | **Vi** |
| 7 | **Pa** |
| 8 | **Sn** |
| 9 | **Lu** |
| 10 | **Li** |
| 11 | **Sc** |

Table 1.2: A solution to study group accommodation problem

A natural question would be to find how the students should be allocated apartments such that each study group has the *least distance to travel* for a discussion? More specifically, we are interested in enforcing additional conditions, namely, that all the students in a study group must be next to each other; in other words, for one student to reach another fellow study group member's apartment (for all study groups the student is part of), she must not have to pass the apartment of any student who is not in that study group. To further elucidate, the apartments of students of any study group must be arranged in an exclusive unfragmented path on the street map. Exclusivity here means that the path must not have apartments from other study groups (unless that apartment is also part of *this* study group).

An intuitive approach to this problem would be to first find the paths that each study group decides to inhabit and then refine the allocation to individual students. A feasible

allocation of exclusive routes to study groups is illustrated in Fig 1.2 [c1] and the students' allocation of apartments that obeys this route allocation is also shown. Table 1.2 shows the same solution set theoretically. How this is algorithmically computed is the focus of this thesis.

[c1] *TH*: fig:streetmappat

[c1]

[c1] *TH*: make dashed/textured lines for routes. make it color agnostic.

As a special case, suppose all the apartments are on the same street or if they are all lined up on a single path, the street map becomes a tree that is just a path. Then the problem becomes what is called an *interval assignment problem*. The idea of interval assignment may not be obvious here; hence to see this, consider a different problem in *Wallace Studies Institute* where the classes for these study groups courses need to be scheduled during a day (or a week or any time period). Each study group has a bunch of courses associated with it some of which may be shared by two or more study groups. It is mandatory that a student who is a member of a study group takes all the courses associated with that group. There are slots during the day for classes to be held and the problem is to allocate class slots to courses such that all the classes of a study group are consecutive. It is debatable if this will not hamper the attention span and memory retention rate of the students but that is, regrettably, out of the scope of this thesis. The parallels between this class allocation problem and the accommodation problem can be seen as follows. The set $U$ here, are the courses offered (say Course 101 *"Influence of post modernism in Wallace's work"*, Course 102 *"A study on fragmented prose method"* and so on). In this variation of the problem, the collection $\mathcal{F}$ is the set of study groups but the study groups are filled by course IDs (in place of students in the earlier example). For instance, Course 101 is mandatory for all study groups $\mathbb{B}$, $\mathbb{T}$, $\mathbb{W}$, $\mathbb{F}$ and Course 102 is mandatory for only the $\mathbb{B}$ group) and so on. The sequence of class slots for the day (or week or any time period) is analogous to the street map in the accommodation problem. It is quite obvious now why this version of the problem (where the "target graph" is a path and not any tree[c2]) is called an interval assignment problem.

[c2] *TH*: Allowing any tree in this example could be seen as a scenario where there are parallel classes. A node falling in the path between two other nodes would mean that the corresponding is scheduled between the other two.

The interval assignment problem to a set system is equivalent to the consecutive-ones property (COP) problem in binary matrices[Hsu02, NS09]. The COP problem is to rearrange rows (columns) of a binary matrix in such a way that every column (row)

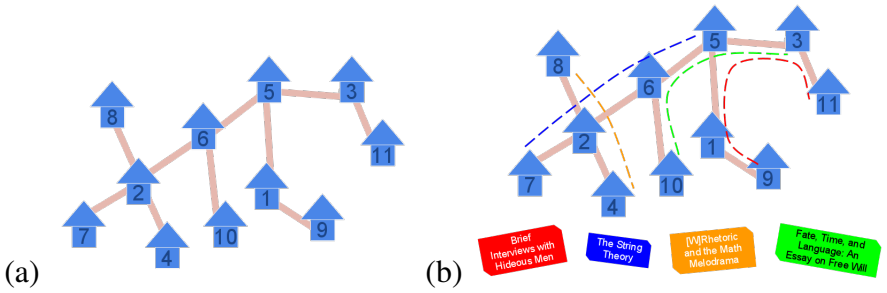



Figure 1.1: (a) *Infinite Loop* street map (b) *Infinite Loop* street map with study group routes allocated. Routes are color coded as follows: red for $\mathbb{B}$ group, blue for $\mathbb{T}$ group, orange for $\mathbb{W}$ group, green for $\mathbb{F}$ group

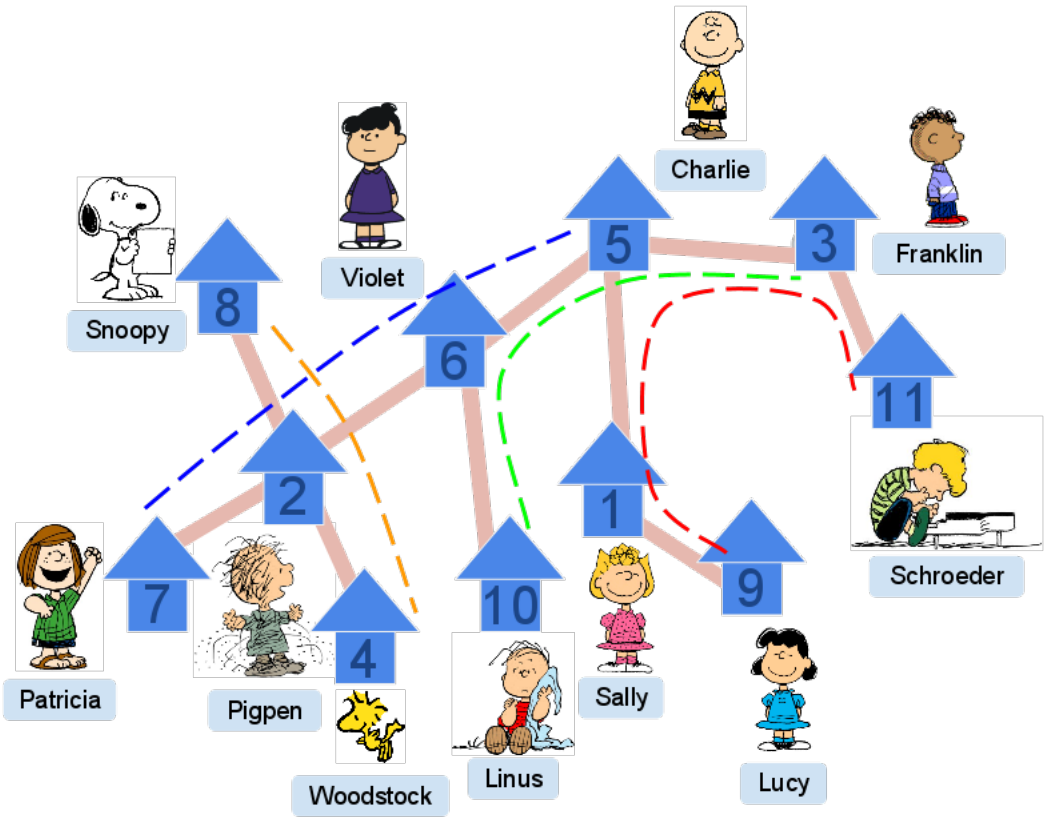



Figure 1.2: Individual allocation of apartments to students in *Infinite Loop* that meets the requirements stated before. The routes are color coded as follows: red for $\mathbb{B}$ group, blue for $\mathbb{T}$ group, orange for $\mathbb{W}$ group, green for $\mathbb{F}$ group. *Peanuts images © Charles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

has its **1**s occur consecutively. If this is possible the matrix is said to have the COP. COP is a well researched combinatorial problem and has several positive results on tests for it and computing the COP permutation (i.e. the course schedule in the above illustration) which will be surveyed later in this document. Hence we are interested in extensions of COP, more specifically, the extension of interval assignment problem to tree path assignment problem (which is illustrated by the study group accommodation problem).

## 1.2 Basic preliminaries - general definitions and nomenclature

c1

c1 (definitions theorems etc) needed if any - graph theory

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

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

c2 c3 c4c5c6 c7c8c9c10c11c12

c2 *TH*: ADD: As it will be described in detail later in this document, isomorphism of certain

c3 *THX*: ADD: peo exists iff chordal. lexicographic BFS [tag:chordalGraph]

c4 *THX*: ADD: A well known result in

c5 *THX*: verify from paper the statement of claim.

c6 *TH*: maximal clique vertex incidence matrix of

c7 *THX*: citation?!!

c8 *TH*: cite fg65 uses these results to give the first polynomial time algorithm for COT.

c9 *THX*: check. how do they use it?

c10 *TH*: A bipartite graph is convex

c11 *THX*: the latter being Tucker's?

c12 *THX*: (2) TBD survey –

### 1.3.1 Matrices with COP

As seen earlier, the interval assignment problem (illustrated as the course scheduling problem in Section 1.1), is a special case of the problem we address in this thesis, namely the tree path labeling problem (illustrated as the study group accommodation problem). The interval assignment problem and COP problem are equivalent problems. In this section we will see some of the results that exists in the literature today towards solving the COP problem and optimization problems surrounding it.

Recall that a matrix with COP is one whose rows (columns) can be rearranged so that the **1**s in every column (row) are in consecutive rows (columns). Figure 1.3 shows examples of this property. COP in binary matrices has several practical applications in diverse fields including scheduling [HL06], information retrieval [Kou77] and computational biology [ABH98]. Further, it is a tool in graph theory [Gol04] for interval graph recognition, characterization of Hamiltonian graphs, planarity testing [BL76] and

$M_1$:

| $c_1$ | $c_2$ | $c_3$ | $c_4$ |
|---|---|---|---|
| **1** | 0 | **1** | 0 |
| 0 | **1** | 0 | **1** |
| **1** | 0 | 0 | **1** |

$M_1'$:

| $c_3$ | $c_1$ | $c_4$ | $c_2$ |
|---|---|---|---|
| **1** | **1** | 0 | 0 |
| 0 | 0 | **1** | **1** |
| 0 | **1** | **1** | 0 |

$M_2$:

| $d_1$ | $d_2$ | $d_3$ | $d_4$ |
|---|---|---|---|
| **1** | **1** | 0 | 0 |
| 0 | **1** | **1** | 0 |
| 0 | **1** | 0 | **1** |

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

in integer linear programming [HT02, HL06].

The obvious first questions after being introduced to the consecutive ones property of binary matrices are if COP can be detected efficiently in a binary matrix and if so, can the COP permutation of the matrix also be computed efficiently? Recognition of COP in a binary matrix is polynomial time solvable and the first such algorithm was given by [FG65]. A landmark result came a few years later when [Tuc72] discovered the families of forbidden submatrices that prevent a matrix from having COP and most, if not all, results that came later were based on this discovery which connected COP in binary matrices to convex bipartite graphs. In fact, the forbidden submatrices came as a corollary to the discovery that convex bipartite graphs are AT-free in [Tuc72][c1]. The first linear time algorithm for COP testing (COT) was invented by [BL76] using a data structure called PQ trees. Since then several COT algorithms have been invented – some of which involved variations of PQ trees [MM96, Hsu01, McC04], some involved set theory and ICPIA [Hsu02, NS09], parallel COT algorithms[AS95, BS03, CY91] and certifying algorithms[McC04].

[c1] *GTC*: check

The construction of PQ trees in [BL76] draws on the close relationship of matrices with COP to interval graphs. A PQ tree of a matrix is one that stores all row (column) permutations of the matrix that give the COP orders (there could be multiple orders of rows or columns) of the matrix. This is constructed using an elaborate linear time procedure and is also a test for planarity[c2]. PQR trees is a generalized data structure based on PQ trees [MM96, MPT98]. [TM05] describes an improved algorithm to build PQR trees. [c3][Hsu02] describes the simpler algorithm for COT. Hsu also invented PC trees [Hsu01][c4] which is claimed to be much easier to implement. [NS09] describes a characterization of consecutive-ones property solely based on the cardinality properties

[c2] *GTC*: check check check. both interval graph and planarity in this paper?

[c3] *GTC*: improv in terms of what?

[c4] *TH*: This result first appeared inproc ISAAC92

of the set representations of the columns (rows); every column (row) is equivalent to a set that has the row (column) indices of the rows (columns) that have one entries in this column (row). This is interesting and relevant, especially to this thesis because it simplifies COT to a great degree. [c5]

[c5] *GTC*: it reduces the solution search space. fill in the blanks.

[McC04] describes a different approach to COT. While all previous COT algorithms gave the COP order if the matrix has the property but exited stating negative if otherwise, this algorithm gives an evidence by way of a certificate of matrix even when it has no COP. This enables a user to verify the algorithm's result even when the answer is negative. This is significant from an implementation perspective because automated program verification is hard and manual verification is more viable. Hence having a certificate reinforces an implementation's credibility. Note that when the matrix *has* COP, the COP order is the certificate. The internal machinery of this algorithm is related to the weighted betweenness problem addressed[c1] in [COR98]. [c2] [c3]

[c1] *GTC*: in what way??

[c2] *GTC*: expand on the COP order graph creation and it having to be bipartite for M to have COP. and thus an odd cycle being an evidence of no COP.

[c3] *TH*: where should this go?: (1) citeljlm97 (application of PQ trees in graphics). (2) helly's theorem citation 19XXdgk-Hellystheorem-Danzer-Gruenbaum-Klee

### 1.3.2 Optimization problems in COP

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

[c4c5] [Tuc72] showed that a matrix that does not have COP have certain substructures that prevent it from having COP. Tucker classified these forbidden substructures into five classes of submatrices. This result is presented in the context of convex bipartite graphs which [Tuc72] proved to be AT-free[c6]. By definition, convex bipartite graph have half adjacency matrices that have COP on either rows or columns (graph is biconvex if it has COP on both)[Dom08]. A half adjacency matrix is a binary matrix representing a bipartite graph as follows. The set of rows and the set of columns form the two partitions of the graph. Each row node is adjacent to those nodes that represent the

[c4] *TH*: – sect 4.1 in cite:d08phd has many results surveyed. hardness results, approx. results. results are usually for a class of matrices $(a, b)$ where number 1s in columns and rows are restriced to $a$ and $b$ . – problem of flipping at most $k$ entries of $M$ to make it attain COP. this is NP complete cite:b75-phd

[c5] *TH*: (1) scite:lb62 showed that interval graphs are AT-free. describe AT (2) show the close relationship b/w COP and graphs sec 2.2, pg 31

[c6] *TH*: check this up. give details. - doms'

columns that have **1**s in the corresponding row. [Tuc72] proves that this bipartite graph has no asteroidal triple if and only if the matrix has COP and goes on to identify the forbidden substructures for these bipartite graphs. The matrices corresponding to these substructures are the forbidden submatrices.

Once a matrix has been detected to not have COP (using any of the COT algorithms mentioned earlier), it is naturally of interest to find out the smallest forbidden substructure (in terms of number of rows and/or columns and/or number of entries that are **1**s). [Dom08] discusses a couple of algorithms which are efficient if the number of **1**s in a row is small. This is of significance in the case of sparse matrices where this number is much lesser than the number of columns. $(*, \Delta)$-*matrices* are matrices with no restriction on number of **1**s in any column but has at most $\Delta$ **1**s in any row. MIN COS-R (MIN COS-C), MAX COS-R (MAX COS-C) are similar problems which deals with inducing COP on a matrix. In MIN COS-R (MIN COS-C) the question is to find the minimum number of rows (columns) that must be deleted to result in a matrix with COP. In the dual problem MAX COS-R (MAX COS-C) the search is for the maximum number of rows (columns) that induces a submatrix with COP. Given a matrix $M$ with no COP, [Boo75] shows that finding a submatrix $M'$ with all columns[c1] but a maximum cardinality subset of rows such that $M'$ has COP is NP complete. [HG02] corrects an error of the abridged proof of this reduction as given in [GJ79]. [Dom08] discusses all these problems in detail giving an extensive survey of the previously existing results which are almost exhaustively all approximation results and hardness results. Taking this further, [Dom08] presents new results in the area of parameterized algorithms for this problem[c2].

[c1] *GTC*: check if b75 deals with COP col or COP row. also is it any submatrix with k less than r rows or submatrix must have all columns?

[c2] *TH*: elaborate - what are the results?

Another problem is to find the minimum number of entries in the matrix that can be toggled to result in a matrix with COP. [Vel85] discusses approximation of COP AUGMENTATION which is the problem of changing of the minimum number of zero entries to **1**s so that the resulting matrix has COP. As mentioned earlier, this problem is known to be NP complete due to [Boo75]. [Vel85] also proves, using a reduction to the longest path problem, [c3] that finding a Tucker's forbidden submatrix of at least $k$ rows is NP complete. [c4] [c5]

[c3] *GTC*: or is it a survey of another result? check.

[c4] *GTC*: how is this different from booth's 75 result??

[c5] *TH*: where should this go? citeltz04 (approx submatrix with COP sparse matrices)

[JKC+04] discusses the use of matrices with almost-COP (instead of one block of

consecutive **1**s, they have $x$ blocks, or *runs*, of consecutive **1**s and $x$ is not too large) in the storage of very large databases. The problem is that of reordering of a binary matrix such that the resulting matrix has at most $k$ runs of **1**s. This is proved to be NP hard using a reduction from the Hamiltonian path problem.[c6] [c7] [c8] [c9] [c10]

[c6] *TH*: Theorem 2.1 in jkckv

[c7] *TH*: (1) A connection of COP problem to the travelling salesman problem is also introduced. what does this mean? – COP can be used as a tool to reorder $0.5T \leq runs(M) \leq$ (2) The optimization version of the $k$-run problem, i.e. minimization of number of blocks of ones is proven to be NP complete by cite:k77

[c8] *GTC*: are these two the same?

[c9] *TH*: what is the reduction?

[c10] *TH*: other problems similar to COP – cite:ckl96 (ILP, circ ones, one drop) – cite:th98 (generalization of COP - minimax, biotonic column) Tucker

## 1.4 Generalization of COP - The Motivation

As seen in Section 1.3.1, [NS09] describes a characterization of consecutive-ones property based on the cardinality properties of the set representations of the columns. This result is very relevant to this thesis because aside from it simplifying COT to a great degree, our generalization problem is motivated by their results.

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

The tree path labeling problem is a natural generalization of the interval assignment problem or the COP problem. The problem is defined as follows – given a set system $\mathcal{F}$ from a universe $U$ and a tree $T$, does there exist a bijection from $U$ to the vertices of $T$ such that each set in the system maps to a path in $T$. We refer to this as the *tree path labeling problem* for an input set system, target tree pair – $(\mathcal{F}, T)$. As a special case if the tree $T$ is a path, the problem becomes the interval assignment problem. We focus on the question of generalizing the notion of an ICPIA [NS09] to characterize feasible path assignments. We show that for a given set system $\mathcal{F}$, a tree $T$, and an assignment of paths from $T$ to the sets, there is a feasible bijection between $U$ and $V(T)$ if and only if all intersection cardinalities among any three sets (not necessarily distinct) is same as the intersection cardinality of the paths assigned to them and the input passes

a filtering algorithm (described in this paper) successfully. This characterization gives a natural data structure that stores all the relevant feasible bijections between $U$ and $V(T)$. This reduces the search space for the solution considerably from the universe of all possible bijections between $U$ and $V(T)$ to only those bijections that maintain the characterization[c1]. Further, the filtering algorithm is also an efficient algorithm to test if a tree path labeling to the set system is feasible.

[c1] *TH*: REWORD

## 1.5 Summary of results

As we saw earlier by the result in [NS09], pairwise intersection cardinality preservation is necessary and sufficient for an interval assignment to be feasible for a given hypergraph[c0] and thus is a characterization for COP. In our work we extend this characterization and find that trio-wise intersection cardinality preservation makes a tree path labeling [c0] (TPL) feasible, which is a generalization of the COP problem. This problem is defined as follows.

FEASIBLE TREE PATH LABELING

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

We give a necessary and sufficient condition (called ICPPL)[c1] for FEASIBLE TREE PATH LABELING to output in affirmative. This characterization can be checked in polynomial time. The most interesting consequence is that in our constructive procedure, it is sufficient to iteratively check if three-way intersection cardinalities are preserved. In other words, in each iteration, it is sufficient to check if the intersection of any three hyperedges is of the same cardinality as the intersection of the corresponding paths. Thus this generalizes the well studied question of the feasible interval assignment problem which is the special case when the target tree $T$ is simply a path [Hsu02, NS09].

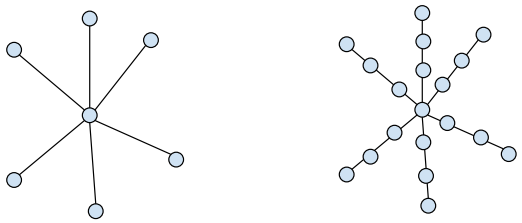
[c1] *GTC*: say a few words about icppl

[c2]

[c2] A few lines on the algo itself.

[c0] A *hypergraph* is an alternate representation of a set system and will be used through out this section.

[c0] A *tree path labeling* $\ell$ is a bijection of paths from the target tree $T$ to the hyperedges in given hypergraph $\mathcal{F}$.



(a) (b)

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

Aside from checking if a given TPL is feasible, we also solve the problem of computing a feasible TPL for a given hypergraph and target tree, if one exists. This problem, COMPUTE FEASIBLE PATH LABELING, is defined as follows.

COMPUTE FEASIBLE PATH LABELING

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

We present a polynomial time algorithm for COMPUTE FEASIBLE PATH LABELING when the target tree $T$ belongs to a special class of trees called *$k$-subdivided stars* and when the hyperedges in the hypergraph $\mathcal{F}$ have at most $k + 2$ vertices. A couple of examples of $k$-subdivided stars are given in Figure 1.4.

COMPUTE $k$-SUBDIVIDED STAR PATH LABELING

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

In spite of this being a restricted case, we believe that our results are of significant interest in understanding the nature of GRAPH ISOMORPHISM which is polynomial time solvable in interval graphs while being hard on path graphs[KKLV10]. $k$-subdivided stars are a class of trees which are in many ways very close to intervals or paths. Each ray[c0] are independent except for the root[c0] and hence can be considered as an independent interval till the root. Our algorithm builds on this fact and uses the interval

[c0]The path from a leaf to the root, the vertex with highest degree, is called a *ray* of the $k$-subdivided star.

[c0]The vertex with maximum degree in a $k$-subdivided star is called *root*

assignment algorithm[NS09] up until "reaching" the root and then uses the trio-wise intersection cardinality (the extra condition in ICPPL after ICPIA) check to resolve the ambiguity about which ray the algorithm should "grow" the solution into in the next iteration.

We also have an algorithm for solving COMPUTE FEASIBLE PATH LABELING with no restrictions which runs in exponential time. This algorithm finds a path labeling from $T$ by decomposing the problem into smaller subproblems of finding path labeling of subsets of $\mathcal{F}$ from subtrees of $T$. Given the fact that binary matrices naturally represent a set system and that the overlap relation between the sets involved is an obvious equivalence relation, $\mathcal{F}$ naturally partitions into equivalence classes known as overlap components. In the context of COP, *overlap components* were used in [Hsu02] and [KKLV10]. Moreover, [NS09] discovered that these equivalence classes form a total order. We extend this to TPL and find that when $\mathcal{F}$ is a path hypergraph[c0], the classes can be partially ordered as an in-tree in polynomial time. Once $\mathcal{F}$ is "broken" into overlap components, one must identify the subtree of $T$ that it needs to map to and this is the hard part which is currently open to be solved in polynomial time.

c1

c1 *TH*: The connection of TPL to graph isomorphism will be made later in the document

[c0] If there exists an FTPL for a hypergraph $\mathcal{F}$, it is called a path hypergraph.

# CHAPTER 2

# Consecutive-Ones property - Important Results from Literature

c1

c2

## 2.1 Matrices with COP

c3

## 2.2 Optimization problems in COP

c4

c1 TBD: have a few lines about organization of chapter

c2 Survey chapter: is the full fledged expansion of the survey in introduction with details, observations, theorems etc.

c3 Expand on ref:sec:copmatrices

c4 Expand on ref:sec:optcop

# CHAPTER 3

# Preliminaries to new results

c1

c1 TBD: have a few lines about organization of chapter

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

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

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

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

Two hypergraphs $\mathcal{F}'$, $\mathcal{F}''$ are said to be *isomorphic* to each other, denoted by $\mathcal{F}' \cong \mathcal{F}''$, iff there exists a bijection $\phi : supp(\mathcal{F}') \to supp(\mathcal{F}'')$ 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]. This is called *hypergraph isomorphism*. c2

c2 also extend $\phi$ to hyperedges – see if required

The *intersection graph* $\mathbb{I}(\mathcal{F})$ of a hypergraph $\mathcal{F}$ is a graph such that its vertex set has a bijection to $\mathcal{F}$ and there exists an edge between two vertices iff their corresponding hyperedges have a non-empty intersection [Gol04].

If the intersection graphs of two hypergraphs are isomorphic, $\mathbb{I}(\mathcal{F}) \cong \mathbb{I}(\mathcal{P})$ where $\mathcal{P}$ is also a path system, then the bijection $\ell : \mathcal{F} \to \mathcal{P}$ due to this isomorphism is called a *path labeling* of the hypergraph $\mathcal{F}$. 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 $\ell$ is defined as follows: $\ell(S_1) = P_1$ iff $g(v_{S_1}) = v_{P_1}$. The path system $\mathcal{P}$ may be alternatively denoted in terms of $\mathcal{F}$ and $\ell$ as $\mathcal{F}^{\ell}$. In most scenarios in this paper, what is given are the pair $(\mathcal{F}, \ell)$ and the target tree $T$; hence this notation will be used more often.

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

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

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

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

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

# CHAPTER 4

# Tree Path Labeling of Path Hypergraphs - the New Results

[c1] The problem of COP testing can be easily seen as a simple constraint satisfaction problem involving a system of sets from a universe. Every column of the binary matrix can be converted into a set of integers which are the indices of the rows with 1s in that column. When observed in this context, if the matrix has the COP, a reordering of its rows will result in sets that have only consecutive integers. In other words, the sets after reordering are intervals. Indeed the problem now becomes finding interval assignments to the given set system [NS09] with a single permutation of the universe (set of row indices) which permutes each set to its interval. The result in [NS09] characterizes interval assignments to the sets which can be obtained from a single permutation of the rows. They show that for each set, the cardinality of the interval assigned to it must be same as the cardinality of the set, and the intersection cardinality of any two sets must be same as the interesction cardinality of the corresponding intervals. While this is naturally a necessary condition, [NS09] shows this is indeed sufficient. Such an interval assignment is called an Intersection Cardinality Preserving Interval Assignment (ICPIA). Finally, the idea of decomposing a given 0-1 matrix into prime matrices to check for COP is adopted from [Hsu02] to test if an ICPIA exists for a given set system.

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

[c2] The necessary preliminaries with definitions etc. are presented in Section 3. Section 4.2 documents the characterization of a feasible path labeling and finally, Section 4.3.1 describes a polynomial time algorithm to find the tree path labeling of a given set system from a given $k$-subdivided tree.

c2 TBD: have a few lines about organization of chapter

## 4.1 Introduction

A natural generalization of the interval assignment problem is feasible tree path labeling problem of a set system. The problem is defined as follows – given a set system $\mathcal{F}$ from a universe $U$ and a tree $T$, does there exist a bijection from $U$ to the vertices of $T$

such that each set in the system maps to a path in $T$. We refer to this as the *tree path labeling problem* for an input set system, target tree pair – $(\mathcal{F}, T)$. As a special case if the tree $T$ is a path, the problem becomes the interval assignment problem. We focus on the question of generalizing the notion of an ICPIA [NS09] to characterize feasible path assignments. We show that for a given set system $\mathcal{F}$, a tree $T$, and an assignment of paths from $T$ to the sets, there is a feasible bijection between $U$ and $V(T)$ if and only if all intersection cardinalities among any three sets (not necessarily distinct) is same as the intersection cardinality of the paths assigned to them and the input runs a filtering algorithm (described in this paper) successfully. This characterization is proved constructively and it gives a natural data structure that stores all the relevant feasible bijections between $U$ and $V(T)$. Further, the filtering algorithm is also an efficient algorithm to test if a tree path labeling to the set system is feasible. This generalizes the result in [NS09].

[c1]It is an interesting fact that for a matrix with the COP, the intersection graph of the corresponding set system is an interval graph. A similar connection to a subclass of chordal graphs and a superclass of interval graphs exists for the generalization of COP. In this case, the intersection graph of the corresponding set system must be a *path graph*. Chordal graphs are of great significance, extensively studied, and have several applications. One of the well known and interesting properties of a chordal graphs is its connection with intersection graphs [Gol04]. For every chordal graph, there exists a tree and a family of subtrees of this tree such that the intersection graph of this family is isomorphic to the chordal graph [Ren70, Gav78, BP92]. These trees when in a certain format, are called *clique trees* [PPY94] of the chordal graph. This is a compact representation of the chordal graph. There has also been work done on the generalization of clique trees to clique hypergraphs [KM02]. If the chordal graph can be represented as the intersection graph of paths in a tree, then the graph is called path graph [Gol04]. Therefore, it is clear that if there is a bijection from $U$ to $V(T)$ such that for every set, the elements in it map to vertices of a unique path in $T$, then the intersection graph of the set system is indeed a path graph. However, this is only a necessary condition and can be checked efficiently because path graph recognition is polynomial time solvable[Gav78, Sch93]. Indeed, it is possible to construct a set system and tree, such that the intersection graph is a path graph, but there is no bijection

c1 the following text could be in Further Study/Conclusion

between $U$ and $V(T)$ such that the sets map to paths. Path graph isomorphism is known be isomorphism-complete, see for example [KKLV10]. An interesting area of research would be to see what this result tells us about the complexity of the tree path labeling problem (not covered in this paper).

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

## 4.2 Characterization of Feasible Tree Path Labeling

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

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

The following lemma is useful in subsequent arguments.

**Lemma 4.2.1** *If $\ell$ is an ICPPL, and $S_1, S_2, S_3 \in \mathcal{F}$, then* $\lvert S_1 \cap (S_2 \setminus S_3)\rvert = \lvert\ell(S_1) \cap (\ell(S_2) \setminus \ell(S_3))\rvert$.

**Proof** Let $P_i = \ell(S_i)$, for all $1 \leq i \leq 3$. $\lvert S_1 \cap (S_2 \setminus S_3)\rvert = \lvert(S_1 \cap S_2) \setminus S_3\rvert = \lvert S_1 \cap S_2\rvert - \lvert S_1 \cap S_2 \cap S_3\rvert$. Due to properties (ii) and (iii) of ICPPL, $\lvert S_1 \cap S_2\rvert - \lvert S_1 \cap S_2 \cap S_3\rvert = \lvert P_1 \cap P_2\rvert - \lvert P_1 \cap P_2 \cap P_3\rvert = \lvert(P_1 \cap P_2) \setminus P_3\rvert = \lvert P_1 \cap (P_2 \setminus P_3)\rvert$. 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` $(\mathcal{F}, \ell, T)$

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

**Lemma 4.2.2** *In Algorithm 1, if input $(\mathcal{F}, \ell)$ is a feasible path assignment then at the end of $j$th iteration of the* **while** *loop, $j \geq 0$, $(\mathcal{F}_j, \ell_j)$ is a feasible path assignment.*

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

$\ell_{\phi[\mathcal{F}_{j-1}]}(S_1) \setminus \ell_{\phi[\mathcal{F}_{j-1}]}(S_2)$. Thus the induced path labeling $\ell_{\phi[\mathcal{F}_j]} = \ell_j$. Therefore lemma is proven. ∎

**Lemma 4.2.3** *In Algorithm 1, at the end of $j$th iteration, $j \geq 0$, of the* **while** *loop, the following invariants are maintained.*

| | | |
|---|---|---|
| *I* | $\ell_j(R)$ *is a path in* $T$, | *for all* $R \in \mathcal{F}_j$ |
| *II* | $\lvert R\rvert = \lvert\ell_j(R)\rvert$, | *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 $j$th iteration in line 4 of Algorithm 1, $S_1 \cap S_2, S_1 \setminus S_2, S_2 \setminus S_1$ and its images in $\ell_j$ where $\ell_{j-1}(S_1)$ and $\ell_{j-1}(S_2)$ intersect and share a leaf.

The invariants are true in the base case $(\mathcal{F}_0, \ell_0)$, since it is the input ICPPL. Assume the lemma is true till the $j-1$th iteration. Let us consider the possible cases for each of the above invariants for the $j$th iteration.

✠ *Invariant* I/II

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

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

✠ *Invariant* III

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

IIIb | *Only one, say* $R$*, is a new set.* Due to invariant IV induction hypothesis, Lemma 4.2.1 and definition of $\ell_j$, it follows that invariant III is true no matter which of the new sets $R$ is equal to. If $R = S_1 \cap S_2$, $|R \cap R'| = |S_1 \cap S_2 \cap R'| = |\ell_{j-1}(S_1) \cap \ell_{j-1}(S_2) \cap \ell_{j-1}(R')| = |\ell_j(S_1 \cap S_2) \cap \ell_j(R')| = |\ell_j(R) \cap \ell_j(R')|$. If

$R = S_1 \setminus S_2$, $|R \cap R'| = |(S_1 \setminus S_2) \cap R'| = |(\ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2)) \cap \ell_{j-1}(R')| = |\ell_j(S_1 \cap S_2) \cap \ell_j(R')| = |\ell_j(R) \cap \ell_j(R')|$. Similarly, if $R = S_2 \setminus S_1$. Note $R'$ is not a new set.

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

✠ *Invariant* IV

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

∎

**Lemma 4.2.4** *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 4.2.2 $(\mathcal{F}_j, \ell_j)$ for every iteration $j > 0$ is feasible. Also, every hypergraph isomorphism $\phi : supp(\mathcal{F}) \to V(T)$ that induces $\ell$ on $\mathcal{F}$ also induces $\ell_j$ on $\mathcal{F}_j$, i.e., $\ell_{\phi[\mathcal{F}_j]} = \ell_j$. Thus it can be seen that for all $x \in supp(\mathcal{F})$, for all $v \in V(T)$, if $(x, v) \in \phi$ then $v \in \ell_j(S)$ for all $S \in \mathcal{F}_j$ such that $x \in S$. In other words, filter 1 outputs a filtered path labeling that "preserves" hypergraph isomorphisms of the original path labeling.

Secondly, line 10 will execute iff the exit condition in line 9, i.e. failure of three way intersection preservation, becomes true in any iteration of the **while** loop. Due to Lemma 4.2.3 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 $\ell_{new}$ such that $\ell_{new}(\{x\}) = \{v\}$ where $x$ an arbitrary element from $S \setminus \bigcup_{\hat{S} \neq S} \hat{S}$. In other words, $x$ is an element present in no other set in $\mathcal{F}$ except $S$. This is intuitive since $v$ is present in no other path image under $\ell$ other than $\ell(S)$. The element $x$ and leaf $v$ are then removed from the set $S$ and path $\ell(S)$ respectively. After doing this for all leaves in $T$, all path images in the new path

labeling $\ell_{new}$ except leaf labels (a path that has only a leaf is called the *leaf label* for the corresponding single element hyperedge or set) are paths from a new pruned tree $T_0 = T \setminus \{v \mid v \text{ is a leaf in } T\}$. Algorithm 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: $\quad \mathcal{F}_j \leftarrow \mathcal{F}_{j-1} \setminus \{S_1\}$
5: $\quad$ for all $S \in \mathcal{F}_{j-1}$ such that $S \neq S_1$ set $\ell_j(S) \leftarrow \ell_{j-1}(S)$
6: $\quad X \leftarrow S_1 \setminus \bigcup_{S \in \mathcal{F}_{j-1}, S \neq S_1} S$
7: $\quad$ **if** $X$ is empty **then**
8: $\quad\quad$ **exit**
9: $\quad$ **end if**
10: $\quad x \leftarrow$ arbitrary element from $X$
11: $\quad \mathcal{F}_j \leftarrow \mathcal{F}_j \cup \{\{x\}, S_1 \setminus \{x\}\}$
12: $\quad \ell_j(\{x\}) \leftarrow \{v\}$
13: $\quad \ell_j(S_1 \setminus \{x\}) \leftarrow \ell_{j-1}(S_1) \setminus \{v\}$
14: $\quad j \leftarrow j + 1$
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.

**Lemma 4.2.5** *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 4.2.6** *In Algorithm 2, for all $j > 0$, at the end of the $j$th iteration of the* **while** *loop the four invariants given in Lemma 4.2.3 hold.*

**Proof** By Lemma 4.2.5 we know that set $X$ will not be empty in any iteration of the **while** loop if input ICPPL $(\mathcal{F}, \ell)$ is feasible and $\ell_j$ is always computed for all $j > 0$. Also note that removing a leaf from any path keeps the new path connected. Thus invariant I is obviously true. In every iteration $j > 0$, we remove exactly one element $x$ from one set $S$ in $\mathcal{F}$ and exactly one vertex $v$ which is a leaf from one path $\ell_{j-1}(S)$ in $T$. This is because as seen in Lemma 4.2.5, $x$ is exclusive to $S$ and $v$ is exclusive to $\ell_{j-1}(S)$. Due to this fact, it is clear that the intersection cardinality equations do not change, i.e., invariants II, III, IV remain true. On the other hand, if the input ICPPL is not feasible the invariants are vacuously true. ∎

We have seen two filtering algorithms above, namely, Algorithm 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 $\phi$ 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 $\ell_2$ and the corresponding elements are removed from the corresponding sets in $\mathcal{F}_2$. This tree pruning algorithm is recursively called on the altered hypergraph $\mathcal{F}'$, path label $\ell'$ and tree $T'$. The recursive call returns the bijection $\phi''$ for the rest of the elements in $supp(\mathcal{F})$ which along with the leaf labels $\phi'$ gives us the hypergraph isomorphism $\phi$. The following lemma formalizes the characeterization of feasible path labeling.

**Lemma 4.2.7** *If $(\mathcal{F}, \ell)$ is an ICPPL from a tree $T$ and Algorithm 3,* `get-hypergraph-isomorphi` *$(\mathcal{F}, \ell, T)$ returns a non-empty function, then there exists a hypergraph isomorphism $\phi : supp(\mathcal{F}) \to V(T)$ such that the $\phi$-induced tree path labeling is equal to $\ell$ or $\ell_\phi = \ell$.*

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

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

**if** $T$ is empty **then**
  **return** $\emptyset$
**end if**
$L \leftarrow \{v \mid v \text{ is a leaf in } T\}$
$(\mathcal{F}_1, \ell_1) \leftarrow$ `filter_1` $(\mathcal{F}, \ell, T)$
$(\mathcal{F}_2, \ell_2) \leftarrow$ `filter_2` $(\mathcal{F}_1, \ell_1, T)$
$(\mathcal{F}', \ell') \leftarrow (\mathcal{F}_2, \ell_2)$
$\phi' \leftarrow \emptyset$
**for** every $v \in L$ **do**
  $\phi'(x) \leftarrow v$ where $x \in \ell_2^{-1}(\{v\})$ | `Copy the leaf labels to a one to one function` $\phi' : supp(\mathcal{F}) \rightarrow L$
  Remove $\{x\}$ and $\{v\}$ from $\mathcal{F}'$, $\ell'$ appropriately
**end for**
$T' \leftarrow T \setminus L$
$\phi'' \leftarrow$ `get-hypergraph-isomorphism` $(\mathcal{F}', \ell', T')$
$\phi \leftarrow \phi'' \cup \phi'$
**return** $\phi$

Moreover, by Lemma 4.2.4 and Lemma 4.2.5 it is consistent with the tree path labeling $\ell$ passed. The tree pruning is done by only removing leaves in each call to the function and is done till the tree becomes empty. Thus the returned function $\phi : supp(\mathcal{F}) \rightarrow V(T)$ is a union of mutually exclusive one to one functions exhausting all vertices of the tree. In other words, it is a bijection from $supp(\mathcal{F})$ to $V(T)$ inducing the given path labeling $\ell$ and thus a hypergraph isomorphism. ∎

**Theorem 4.2.8** *A path labeling $(\mathcal{F}, \ell)$ on tree $T$ is feasible iff it is an ICPPL and Algorithm 3 with $(\mathcal{F}, \ell, T)$ as input returns a non-empty function.*

**Proof** From Lemma 4.2.7, we know that if $(\mathcal{F}, \ell)$ is an ICPPL and Algorithm 3 with $(\mathcal{F}, \ell, T)$ as input returns a non-empty function, $(\mathcal{F}, \ell)$ is feasible. Now consider the case where $(\mathcal{F}, \ell)$ is feasible. i.e. there exists a hypergraph isomorphism $\phi$ such that $\ell_\phi = \ell$. Lemma 4.2.4 and Lemma 4.2.5 show us that filter 1 and filter 2 do not exit if input is feasible. Thus Algorithm 3 returns a non-empty function. ∎

### 4.2.1 ICPPL when given tree is a path

Consider a special case of ICPPL with the following properties.

1. Given tree $T$ is a path. Hence, all path labels are interval labels.

2. Only pairwise intersection cardinality preservation is sufficient. i.e. property (iii) in ICPPL is not enforced.

3. The filter algorithms do not have **exit** statements.

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

## 4.3 TPL with special target trees

c1

c1 give problem definition etc

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

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

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

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

I. **Step 1**

(a) **There are type 1 edges, $|\mathcal{T}_1^i| > 0$:** If there is only one $\mathcal{T}_1^i$ hyperedge, label it to the unique path in $R_i$ of length $s(X)$ starting at $v_{i(k+2)} = r$. If there are more than one $\mathcal{T}_1^i$ hyperedges, for any pair of hyperedges $S_1, S_2 \in \mathcal{T}_1^i$, one of the following is true. Let $\ell'(X) \subseteq R_j$ denote the subpath of length $p_1(X)$ that has been assigned to $X$ from a previously considered ray $R_j$.

*Case* 1. $|\ell'(S_1) \cap \ell'(S_2)| = 1$ and $|S_1 \cap S_2| > 1$ i.e. they are in different rays and their residues must contain their intersection. Hence assign both to $R_i$.

*Case* 2. $|\ell'(S_1) \cap \ell'(S_2)| > 1$ and $|S_1 \cap S_2| > |\ell'(S_1) \cap \ell'(S_2)|$ i.e. they are in the same rays and their residues must contain their intersection. Hence assign both to $R_i$.

*Case* 3. $|\ell'(S_1) \cap \ell'(S_2)| > 1$ and $|S_1 \cap S_2| = |\ell'(S_1) \cap \ell'(S_2)|$. i.e. the intersection cardinality has been met. Hence assign only $S_1$ to $R_i$ and $S_2$ must be assigned to some $i \geq p$.

Remove the labeled edge or edges from $\mathcal{O}_i$ and add it to $\mathcal{O}_{i+1}$. Also do not add it to $\mathcal{T}_1^{i+1}$ and add to $\mathcal{L}_2^{i+1}$.

(b) **There are no type-1 edges nor super-marginal type-2 hyperedges:** If all type-2 edges of length at most $k+1$, EXIT by saying there is no ICPPL. Otherwise, Find an $X$ of type-2 such that $|X| \geq k+2$, assign it the unique path starting at $v_{i(k+1)} = r$ of length $k+2$, and set $p_1(X) = k+2, s(X) = |X| - (k+2)$, assign it the unique path starting at $v_{i(k+1)} = r$ of length $k+2$, mark this as a type-1 hyperedge, and add it to $\mathcal{O}_{i+1}$ after removing it from $\mathcal{O}_i$.

(c) **There are super-marginal type-2 hyperedges and no type-1 edges:** Let $X$ be a super-marginal hyperedge. If $|X| \leq k+2$, then $X$ is assigned the path of length $|X|$ starting at $v_{i1}$, and $p_1(X) = |X|, p_2(X) = 0$. $X$ is removed from $\mathcal{O}_i$ and not added to $\mathcal{O}_{i+1}$. If $|X| > k+2$, then $p_1(X) = k+2$, assign it the unique path starting at $v_{i(k+1)} = r$ of length $k+2$ and $s(X) = |X| - (k+2)$. In this case, $X$ is classified as a Type-1 edge, removed from $\mathcal{O}_i$, and added to $\mathcal{O}_{i+1}$.

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

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

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

c1 In the following lemmas we identify a set of necessary conditions for $\mathcal{F}$ to have an ICPPL in the $k$-subdivided star $T$. If during the execution of the above algorithm, one of these necessary conditions is not satisfied, the algorithm exits reporting the non-existence of an ICPPL.

c1 need to formalize the below properties

**Lemma 4.3.1** *Let all hyperedges in $\mathcal{O}_i$ be type-1 edges. Then there is a maximal subset $\mathcal{T}_i \subseteq \mathcal{O}_i$ with the following properties:*

1. *$\mathcal{T}_i$ form an inclusion chain.*

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

**Lemma 4.3.2** *If there are no super-marginal type-2 edges in $\mathcal{O}_i$, then there exists at least $p - i$ type-2 hyperedges $X \in \mathcal{O}_i$ such that $|X| \geq k+2$.*

**Lemma 4.3.3** *At the end of Step-1 in the $i$-th iteration, if one hyperedge $X$ of type-2 is such that $\ell(X) \subseteq R_i$, then all other hyperedges in $\mathcal{O}_i$ are connected to $X$ in the overlap component.*

**Lemma 4.3.4** *At the end of Step-2, If control has exit at any time, there is no ICPPL. If control has not exit, then $R_i$ is saturated. No hyperedge of $\mathcal{O}_{i+1}$ will get a path from $R_i$ in the future iterations. No type-2 hyperedge of $\mathcal{O}_{i+1}$ will get a path from $R_i$. Basically $R_i$ is done.*

**Lemma 4.3.5** *Finally, we need to prove that the assignment is an ICPPL. Secondly, if there is a permutation then maps sets to paths, then it is indeed an ICPPL, and our algorithm will basically find it. It is a unique assignment upto permutation of the leaves.*

**Lemma 4.3.6** *If $X \in \mathcal{F}$ is super-marginal and $(\mathcal{F}, \ell)$ is a feasible tree path labeling to tree $T$, then $\ell(X)$ will contain a leaf in $T$.*
*On the otherhand, if $\ell(X)$ has a leaf in $T$, $X$ is marginal but may not be super-marginal.*

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

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

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

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

## 4.4 TPL with no restrictions

c1

c1
ADD
WALCOM PAPER
TEXT

# CHAPTER 5

# Conclusion

We give a characterization for feasible tree path labeling of path hypergraphs. The proof for this is constructive and computes a feasibility bijection mapping vertices of the hypergraph to vertices of the given target tree. This thesis also discovered an exponential algorithm that computes a feasible TPL to a given hypergraph if it is a path hypergraph.

Our results have close connections to recognition of path graphs and to path graph isomorphism. Graphs which can be represented as the intersection graph of paths in a tree are called *path graphs*[Gol04]. Thus, a hypergraph $\mathcal{F}$ can be interpreted as paths in a tree, if and only if the intersection graph of $\mathcal{F}$ is a *path graph*. Path graphs are a subclass of chordal graphs since chordal graphs are characterized as the intersection graphs of subtrees of a tree[Gol04]. Path graphs are well studied in the literature [Ren70]–[Gol04][c1]. Path graph recognition can be done in polynomial time[Gav78, Sch93]. Clearly, this is a necessary condition in terms of the intersection graph of the input hypergraph $\mathcal{F}$ in FEASIBLE TREE PATH LABELING. However, one can easily obtain a counterexample to show the insufficiency of this condition[c2]. Path graph isomorphism is known be isomorphism-complete[KKLV10]. Therefore, [c3]it is unlikely that we can solve the problem of finding feasible path labeling $\ell$ for a given $\mathcal{F}$ and tree $T$. It is definitely interesting to classify the kinds of trees and hypergraphs for which feasible path labelings can be[c4] found efficiently. [c5]These results would form a natural generalization of COP testing and interval graph isomorphism, culminating in Graph Isomorphism itself. To this effect we consider TPL on $k$-subdivided star and give a polynomial time solution for the same.

Whether TPL on general trees is solvable in $P$ remains open. So do optimization opportunities in TPL (possible extensions to optimization for COP in the Section 1.3.2.[c6]).

c7c8

[c1] *TH*: Check if cite:bp93 is indeed for path graphs

[c2] *GTC*: Give a figure showing insuff.

[c3] *GTC*: WHY?

[c4] *TH*: ELABORATE.

[c5] *TH*: REWORD

[c6] *TH*: link to section

[c7] *TH*: ||Complexity challenges:|| Canonization is an important tool in graph isomorphism. Invention of a

[c8] *TH*: (1) the interesting survey in conclusion section of kklv10 (2) [[[COP helping some problems with hardness]]] having COP in the input structure makes some problems less hard than in a general input. – min set

# APPENDIX A

# Appendix A

# REFERENCES


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

# LIST OF PAPERS BASED ON THESIS

1. Authors.... Title... *Journal*, Volume, Page, (year).