Generalization of the Consecutive-ones Property

A THESIS

submitted by

ANJU SRINIVASAN

for the award of the degree of

MASTER OF SCIENCE *by Research*

from the department of

COMPUTER SCIENCE AND ENGINEERING

at

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Guindy, Chennai - 600036

JANUARY 2012

THESIS CERTIFICATE

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

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

Chennai
31 January 2012

ACKNOWLEDGEMENTS

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

D. F. W.

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

Concrete Mathematics (margin notes)

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

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

L'art c'est la solution au chaos.

(Art is the solution to chaos.)

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

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

New Oxford American Dictionary, 2nd Ed.

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

The Salmon of Doubt, D. N. A.

This is water, this is water.

D. F. W.

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.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

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

NOTATION

2^U Powerset of set U

CHAPTER 1

Introduction

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

1.1 Organization of the document

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

In this chapter, Section 1.2 introduces the main problem of this thesis by way of an illustration. Section 1.3 lays out a few general definitions that are helpful in understanding the rest of the chapter. Section 1.4 gives a brief survey of COP and optimization problems related to it followed by motivation for the thesis in Section 1.6. Section 1.7 presents a summary of our results on the extension of COP namely, the tree path labeling problem.

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

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

1.2 Illustration of the problem

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

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

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

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

unfragmented path on the street map. Exclusivity here means that the path must not have apartments from other study groups (unless that apartment is also part of *this* study group).

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

1.2.1 Special case

As a special case of the study group accommodation problem, suppose all the apartments are on the same street or if they are all lined up on a single path, the street map becomes a tree that is just a path. Then the problem becomes what is called an *interval assignment problem*. The idea of interval assignment may not be obvious here; hence to see this, consider a different problem in *Wallace Studies Institute* where the classes for these study groups courses need to be scheduled during a day (or a week or any time period). Each study group has a bunch of courses associated with it some of which may be shared by two or more study groups. It is mandatory that a student who is a member of a study group takes all the courses associated with that group. There are slots during the day for classes to be held and the problem is to allocate class slots to courses such that all the classes of a study group are consecutive. 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

			Apartment allocation (ϕ)	
T	=	*Street map tree of* Infinite Loop	1	**Sa**
$V(T)$	=	$\{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$	2	**Pi**
$\mathcal{P}$	=	$\{R\mathbb{B}, R\mathbb{T}, R\mathbb{W}, R\mathbb{F}\}$	3	**Fr**
$R\mathbb{B}$	=	$\{9, 1, 5, 3, 11\}$	4	**Wo**
$R\mathbb{T}$	=	$\{7, 2, 6, 5\}$	5	**Ch**
$R\mathbb{W}$	=	$\{8, 2, 4\}$	6	**Vi**
$R\mathbb{F}$	=	$\{10, 6, 5, 3\}$	7	**Pa**
n	=	$\lvert V\rvert = 11$	8	**Sn**
m	=	$\lvert\mathcal{P}\rvert = 4$	9	**Lu**
ℓ	=	*Study group to route mapping*	10	**Li**
$\ell(\mathbb{X})$	=	$R\mathbb{X}$ for all $\mathbb{X} \in \mathcal{F}$	11	**Sc**

Table 1.2: A solution to study group accomodation problem

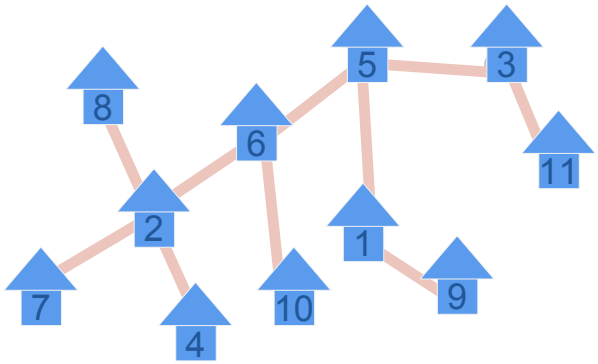


Figure 1.1: *Infinite Loop* street map.

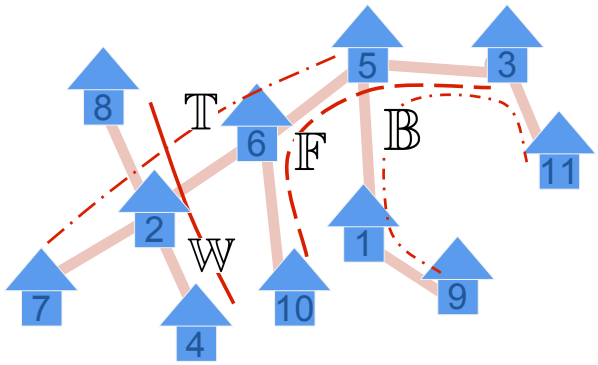


Figure 1.2: *Infinite Loop* street map with study group routes allocated.

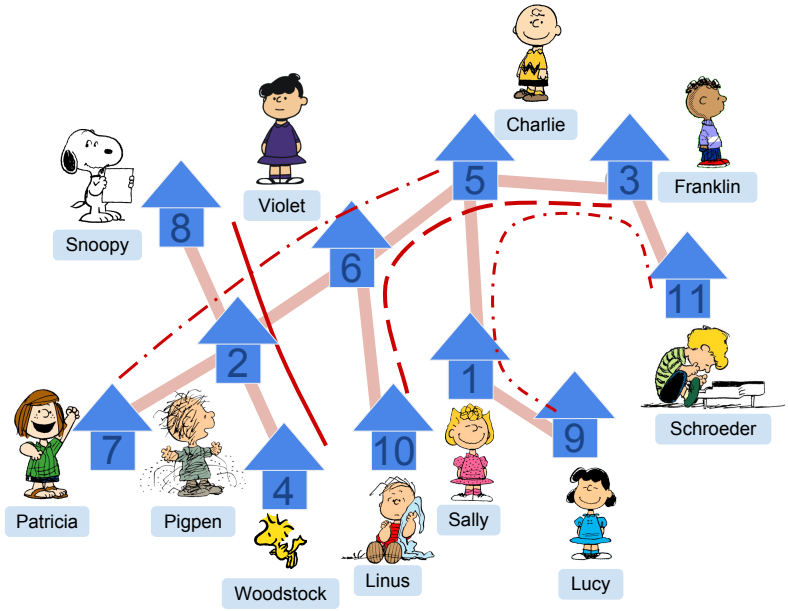


Figure 1.3: Individual allocation of apartments to students in *Infinite Loop* that meets the requirements stated before.

Peanuts images © Charles Schulz

set U here, are the courses offered (say Course 101 *"Influence of post modernism in Wallace's work"*, Course 102 *"A study on fragmented prose method"* and so on). In this variation of the problem, the collection $\mathcal{F}$ is the set of study groups but the study groups are filled by course IDs (in place of students in the earlier example). For instance, Course 101 is mandatory for all study groups $\mathbb{B}$, $\mathbb{T}$, $\mathbb{W}$, $\mathbb{F}$ and Course 102 is mandatory for only the $\mathbb{B}$ group) and so on. The sequence of class slots for the day (or week or any time period) is analogous to the street map in the accommodation problem. It is quite obvious now why this version of the problem (where the "target graph" is a path and not any tree) is called an interval assignment problem.

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

1.3 Basic preliminaries - general definitions and nomenclature

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

Definition 1.3.2. A *permutation* λ of a set $X = \{x_1, x_2, \ldots x_n\}$ is a bijection $\lambda : X \to X$. For simplicity, sometimes, λ is written as a sequence $x_{i_1} x_{i_2} \ldots x_{i_n}$, where $i_j \in \{1, 2 \ldots n\}$ to mean that $\lambda(x_j) = x_{i_j}$.
This document uses both notations as convenient in context.

Definition 1.3.3 (Consecutive-ones property).

1. Consecutive-ones property
2. Strong consecutive-ones property

3. Consecutive-ones property order or consecutive-ones property permutation

4. Consecutive-ones property for set systems

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 1.4: 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].

Definition 1.3.4 (Circular-ones property).

Definition 1.3.5 (set system). also include overlapping sets.

Definition 1.3.6 (Graph, Tree).

Definition 1.3.7 (Maximal clique).

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

1.4 Consecutive-ones Property Testing - a Brief Survey

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

1.4.1 Matrices with COP

As seen earlier, the interval assignment problem (illustrated as the course scheduling problem in Section 1.2), is a special case of the problem we address in this thesis, namely the tree path labeling problem (illustrated as the study group accomodation problem). The interval assignment problem and COP problem are

equivalent problems. In this section we will see some of the results that exists in the literature today towards solving the COP problem and optimization problems surrounding it.

Recall that a matrix with COP is one whose rows (columns) can be rearranged so that the **1**s in every column (row) are in consecutive rows (columns). COP in binary matrices has several practical applications in diverse fields including scheduling [HL06], information retrieval [Kou77] and computational biology [ABH98]. Further, it is a tool in graph theory [Gol04] for interval graph recognition, characterization of Hamiltonian graphs, planarity testing [BL76] and in integer linear programming [HT02, HL06].

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

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

This is interesting and relevant, especially to this thesis because it simplifies COT to a great degree.

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

1.4.2 Optimization problems in COP

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

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

submatrices.

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

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

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

1.5 Application of COP in Areas of Graph Theory and Algorithms

Combine COP in Relational Database Model + COP in Graph Isomorphism + Certifying Algorithms

1.6 Generalization of COP - the Motivation

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

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

The equivalence of the problem of testing for the consecutive-ones property to the constraint statisfaction problem of interval assignment [NS09] or interval labeling [KKLV10] is as follows. Every column (row) of the binary matrix can be converted into a set of non-negative integers which are the indices of rows (columns) with **1**s in that column (row). It is apparent that if the matrix has COP in columns (rows), then constructing such sets after applying the COP permutation to the rows (columns) of the matrix will result in sets with consecutive integers. In other words, after application of COP reordering, the sets are intervals. Indeed the problem now becomes finding interval assignments to a given set system such that there exists a permutation of the universe of set of row indices (column indices) which converts each set to its assigned interval.

The problem of interest in this thesis, namely, tree path labeling problem, is

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

1.7 Summary of New Results in this Thesis

We see in Section 1.6 that pairwise intersection cardinality preservation is necessary and sufficient for an interval assignment to be feasible for a given hypergraph[3] [4] and thus is a characterization for COP [NS09]. In our work we extend this characterization and find that trio-wise intersection cardinality preservation makes a tree path labeling[5] [4] (TPL) feasible, which is a generalization of the COP problem. This problem is defined as follows.

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

[2]The terms *tree path labeling* and *tree path assignment* are, in informal language, synonyms. Formally, the former refers to the bijection $\ell : \mathcal{F} \to \mathcal{P}$. The latter refers to the set of ordered pairs $\{(S, P) \mid S \in \mathcal{F}, P \in \mathcal{P}\}$. $\mathcal{P}$ is a set of paths on T.

[3]A *hypergraph* is an alternate representation of a set system and will be used in this thesis.

[4]See Section 3.2 for the formal definition.

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

Feasible Tree Path Labeling

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

We give a necessary and sufficient condition by way of *Intersection Cardinality Preservation Path Labeling* (ICPPL) and a filtering algorithm for Feasible Tree Path Labeling to output in affirmative. ICPPL captures the trio-wise cardinality property described earlier[6]. This characterization can be checked in polynomial time. A relevant consequence of this constructive procedure is that it is sufficient to iteratively check if three-way intersection cardinalities are preserved. In other words, in each iteration, it is sufficient to check if the intersection of any three hyperedges is of the same cardinality as the intersection of the corresponding paths. Thus this generalizes the well studied question of the feasible interval assignment problem which is the special case when the target tree T is simply a path [Hsu02, NS09].

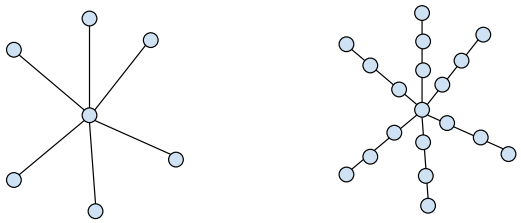
Aside from checking if a given TPL is feasible, we also solve the problem of computing a feasible TPL for a given hypergraph and target tree, if one exists. This problem, Compute Feasible Tree Path Labeling, is defined as follows.

Compute Feasible Tree Path Labeling

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

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

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



(a) (b)

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

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[7] [4] are independent except for the root[8] [4] and hence can be considered as an independent interval till the root. Our algorithm builds on this fact and uses the interval assignment algorithm[NS09] up until "reaching" the root and then uses the trio-wise intersection cardinality (the extra condition in ICPPL that generalizes ICPIA) check to resolve the ambiguity about which ray the algorithm should "grow" the solution into in the next iteration.

We also have an algorithm for solving Compute Feasible Tree Path Labeling with no restrictions on the target tree or set size which runs in exponential time. This algorithm finds a path labeling from T by decomposing the problem into subproblems of finding path labeling of subsets of $\mathcal{F}$ from subtrees of T. Given the fact that binary matrices naturally represent a set system (see Section 1.6) and that the *overlap* relation between the sets involved is an obvious equivalence relation, $\mathcal{F}$ quite naturally partitions into equivalence classes known as *overlap components*. In the context of COP, overlap components were used in [Hsu02]

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

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

and [KKLV10]. Moreover, [NS09] discovered that these equivalence classes form a total order . We extend this to TPL and find that when $\mathcal{F}$ is a path hypergraph[9], the classes can be partially ordered as an in-tree in polynomial time. Once $\mathcal{F}$ is "broken" into overlap components, one must identify the subtree of T that it needs to map to and this is the hard part which is currently open to be solved in polynomial time.

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

CHAPTER 2

Consecutive-ones Property – A Survey of Important Results

This chapter surveys several results that are significant to this thesis or to COP in general. These predominantly pertain to characterizations of COP, algorithmic tests to check for COP (COT), optimization problems on binary matrices that do not have COP and some applications of COP.

2.1 COP in Graph Theory

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

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

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

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

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

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

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

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

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

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

2.1.2–iii. G is an *interval graph* if its vertices can be mapped to intervals on the real line such that two vertices are adjacent if and only if their corresponding intervals overlap . G is an interval graph if and only if its maximal clique matrix has COP [FG65][1]

a. G is a *unit interval graph* if it is an interval graph such that all intervals have the same length.[2]

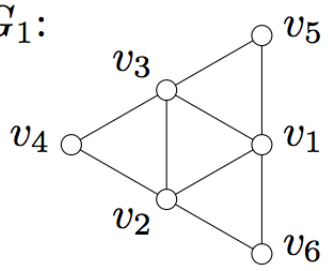
b. G is a *proper interval graph* if it is an interval graph such that no interval properly contains another.[2]

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

[1] This follows [GH64] which states that the maximal cliques of interval graph G can be linearly ordered such that for all $v \in V(G)$, cliques containing v are consecutive in the ordering [Gol04, Th. 8.1].

[2] The set of unit interval graphs and the set of proper interval graphs are the same

G_1:

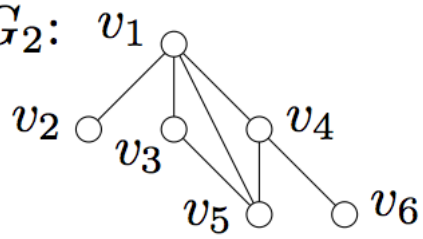


$\mathbf{A_2}$	v_1	v_2	v_3	v_4	v_5	v_6
v_1	**1**	**1**	**1**	0	**1**	**1**
v_2	**1**	**1**	**1**	**1**	0	**1**
v_3	**1**	**1**	**1**	**1**	**1**	0
v_4	0	**1**	**1**	**1**	0	0
v_5	**1**	0	**1**	0	**1**	0
v_6	**1**	**1**	0	0	0	**1**

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

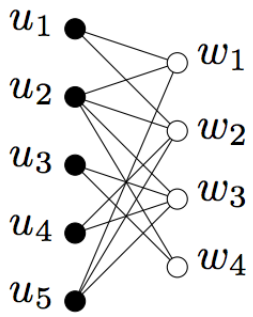
$\mathbf{A'_2}$	v_1	v_6	v_2	v_4	v_3	v_5
v_1	**1**	**1**	**1**	0	**1**	**1**
v_6	**1**	**1**	**1**	0	0	0
v_2	**1**	**1**	**1**	**1**	**1**	0
v_4	0	0	**1**	**1**	**1**	0
v_3	**1**	0	**1**	**1**	**1**	**1**
v_5	**1**	0	0	0	**1**	**1**

G_2:



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

H:



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

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

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

2.1.2–vi. H is *biconvex bipartite* or *doubly convex*[YC95] if its half adjacency matrix has COP on both rows and columns.

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

Interval graphs[3]and circular-arc graphs have a long history in research. The interest around them is due to their very desirable property that several problems that are NP-hard on general graphs, like finding a maximum clique or minimum coloring or independent set, are polynomial time solvable in these graph classes [CLRS01]. In a similar fashion, a lot of problems that are hard on general matrices have efficient solutions on matrices with COP or CROP [Dom08, more citations pg. 33].

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

2.2 Matrices with COP

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

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

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

Graph class	Adjacency matrix	Augmented adjacency matrix	Half adjacency matrix	Maximal clique matrix
Convex-round	⇔ CROP (by defn.)			
Concave-round		⇔ CROP (by defn.)		
⋂				
Circular-arc		⇐ CROP [Tuc72]		⇐ CROP
⋃				
Helly circular-arc		⇐ COP		⇔ CROP [Gav74]
⋃				
Interval		⇐ COP		⇔ COP [FG65]
⋃				
Proper/unit interval		⇔ COP [Rob69]		⇔ COP $r+c$ [Fis85]
		Bipartite graphs		
Circular convex			⇔ CROP (by defn.)	
⋃				
Convex			⇔ COP (by defn.)	
⋃				
Biconvex			⇔ COP $r+c$ (by defn.)	

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

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

a. Does a given binary matrix have COP?

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

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

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

Without doubt, besides computing answers to these questions, we are interested in the efficiency of these computations in terms of computational complexity theory. Results towards questions (a) and (b) are surveyed in this section. Those for question (c) are discussed in Section 2.3 and question (d) is discussed in Section 2.2.3.

It may be noted that one way to design an algorithm to test for COP is by deriving one from any interval graph recognition algorithm using the result HMPV00 [Dom08] which demonstrates how such a derivation can be done. However, this does not necessarily yield an efficient algorithm. We will see results that directly solve the problem on matrices since it is known that questions (a) and (b) stated above for COP are efficiently solvable. Table 2.2 gives a snapshot of these results.

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

Table 2.2: A brief history of COP research

The first polynomial time algorithm for COP testing was by [FG65] which uses overlapping properties of columns with **1**s. Their result has close relations to the characterization of interval graphs by [GH64]. A graph G is an interval graph if and only if all its maximal cliques can be linearly ordered such that for any vertex

v in G, all the cliques that v is incident on are consecutive in this order. Clearly, this means that the maximal clique incidence matrix[4] must have COP on rows.

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

2.2.1 Tucker's forbidden submatrices for COP

[Tuc72] discovered certain forbidden structures for convex bipartite graphs[5] and by definition of this graph class, this translates to a set of forbidden submatrices for matrices with consecutive-ones property. The following are the theorems from [Tuc72] that acheived this characterization.

Theorem 2.2.1 states that convex bipartite graphs cannot have *asteroidal triples*[6] contained in the corresponding vertex partition[7]. Theorem 2.2.2 lists the structures in a bipartite graph that force one of its vertex partitions to have asteriodal triples – in other words, it identifies the subgraphs that prevent the graph from being convex bipartite.

Theorem 2.2.1 ([Tuc72, Th. 6], [Dom08, Th. 2.3]). *A bipartite graph $G = (V_1, V_2, E)$ is convex bipartite on columns*[8] *if and only if V_1 contains no asteroidal triple of G.*

Theorem 2.2.2 ([Tuc72, Th. 7], [Dom08, Th. 2.4]). *In a bipartite graph $G = (V_1, V_2, E)$ the vertex set V_1 contains no asteroidal triple if and only if G contains none of the graphs G_{I_k}, G_{II_k}, G_{III_k} (with $k \geq 1$), G_{IV}, G_V as shown in Figure 2.2 as subgraphs.*

Theorem 2.2.1 and Theorem 2.2.2 result in the following Theorem 2.2.3 which

[4] Definition 2.1.1 iii

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

[6] If $G = (V, E)$ is a graph, a set of three vertices from V form an *asteroidal triple* if between any two of them there exists a path in G that does not contain any vertex from the closed neighborhood of the third vertex.

[7] The partition corresponds to columns (rows) if its half adjacency matrix has COP columns (rows).

[8] Abridged to match terminology adopted in this document. See previous note.

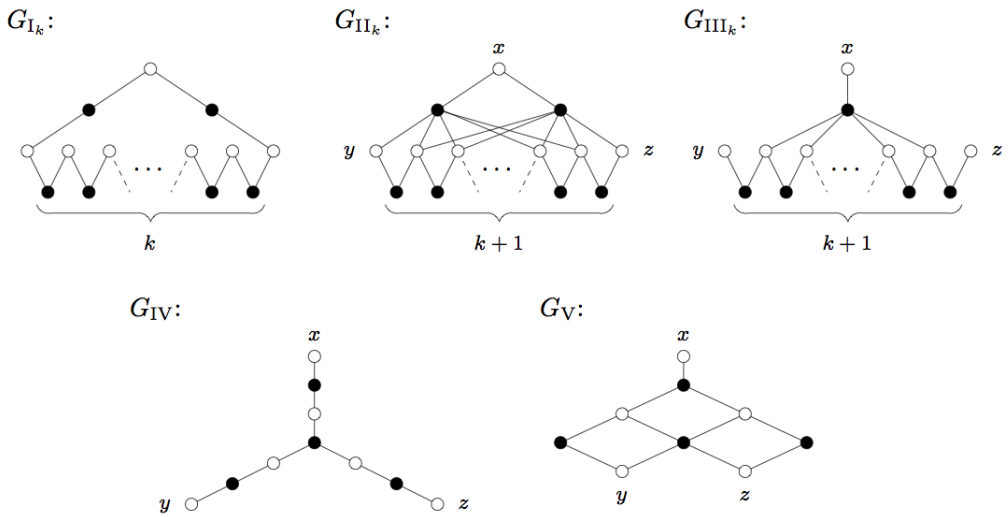


Figure 2.2: Tucker's forbidden subgraphs for convex bipartite graphs. PLACEHOLDER IMG

characterizes matrices with COP.

Theorem 2.2.3 ([Tuc72, Th. 9], [Dom08, Th. 2.5]). *A matrix M has COP if and only if it contains none of the matrices M_{I_k}, M_{II_k}, M_{III_k} (with $k \geq 1$), M_{IV}, M_V as shown in Figure 2.3 as submatrices.*

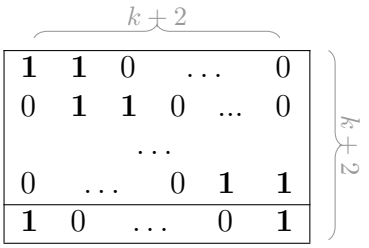
It can be verified that the matrices in Figure 2.3 are the half adjacency matrices of the graphs in Figure 2.2 respectively which is not surprising due to Definition 2.1.2 v.

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

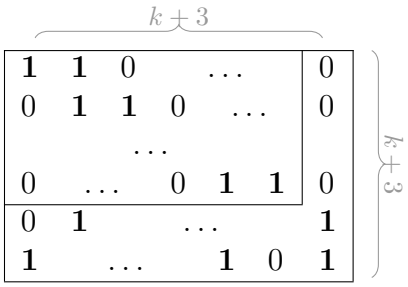
Booth and Lueker in their paper [BL76] gave the first linear algorithm[9] for consecutive-ones property testing while given a linear time interval graph recognition algorithm by a simplification of 's planarity test algorithm. [BL76] introduces a data structure called PQ-tree and their COP testing algorithm is a constructive one that outputs a PQ-tree if the input has COP. A PQ-tree represents all the COP orderings of the matrix it is associated with. [BL76]'s algorithm uses the fact that if a matrix has COP, a PQ-tree for it can be constructed. It is interesting to note that aside from interval graph recognition and COP testing, PQ-tree is also useful

[9]Time complexity is $O(m+n+f)$ where $m \times n$ is the order of the input matrix and f is the number of **1**s in it.

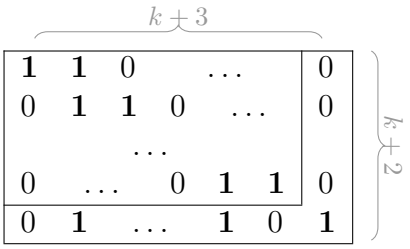
$M_{I_k}, k \geq 1$



$M_{II_k}, k \geq 1$



$M_{III_k}, k \geq 1$



M_{IV}

1	**1**	0	0	0	0
0	0	**1**	**1**	0	0
0	0	0	0	**1**	**1**
1	0	**1**	0	**1**	0

M_V

1	**1**	0	0	0
0	0	**1**	**1**	0
1	**1**	**1**	**1**	0
1	0	**1**	0	**1**

Figure 2.3: Tucker's forbidden submatrices for convex bipartite graphs. [Tuc72]

in other applications like finding planar embeddings of planar graphs [**?**, McC04] and recognizing CROP in a matrix.

Definition 2.2.1 (PQ-tree [BL76, McC04]). A PQ-tree of matrix M with COP on columns (rows), is a tree with the following properties.

i. Each leaf uniquely represents a row (column) of M. The leaf order of the tree gives a COP order for column (row)[10] for M.

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

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

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

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

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

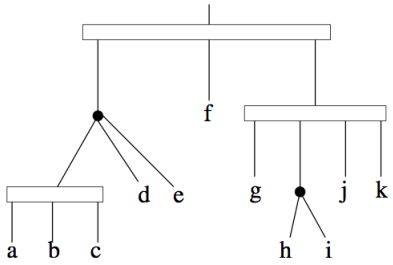


Figure 2.4: An example for PQ-tree. Permuting the order of the left child of the root, we see that (d,a,b,c,e,f,g,h,i,j,k) is a COP order. Reversing the order of the right child of the root, we see that (a,b,c,d,e,f,k,j,h,i,g) is yet another COP order. PLACEHOLDER IMG. [McC04, Fig. 1[1]]

The [BL76] algorithm with input $n \times m$ matrix M starts with a PQ-tree for a vacuous $n \times 0$ matrix M' (submatrix induced by 0 columns). This is known

[10]Note that COP order for column requires permutation of rows and vice versa.

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

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

After the invention of PQ-trees, presumably due to the implementation challenge it posed, there has been several variants of the same in the literature, like PC-tree [SH99, Hsu01, HM03], generalized PQ-tree [KM89, McC04], PQR-tree [MM96, MPT98] etc. Most of these are generalizations of PQ-tree– for instance, PC-tree is generalized to matrices with CROP, PQR-tree and generalized PQ-tree are generalized to matrices and set systems with or without COP. [KM89] invented a modified form of PQ-tree a simpler incremental update of the tree only for recognizing interval graphs. [KR88] constructed efficient parallel algorithms for manipulating PQ trees. dom chapter 2 pg 40 Variations of PQTrees first para - summarize.

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

2.2.3 PQR-tree – COP for set systems

Section 1.6 mentions how a binary matrix naturally maps to a system of sets. A set can be constructed for each column of matrix with its elements being those row indices at which the column has **1**s. Thus the collection of sets corresponding each column of the matrix forms a set system with universe as the set of all row indices of the matrix. This simple construction is formally described in Definition 2.2.2 along with the idea of consecutive-ones property for set systems[11].

[11] As seen in Section 1.2.1

Definition 2.2.2 (Consecutive-ones property for set systems). Let M be a binary matrix of order $n \times m$ and $\{c_i \mid i \in [m]\}$ be the columns in M. A set system $\mathcal{F}_M = \{S_i \mid S_i \subseteq [n], i \in [m]\}$ is defined such that for every column c_i of M, set $S_i = \{j \mid m_{ji} = \mathbf{1}\}$. The collection $\mathcal{F}_M$ is the *set system of binary matrix* M. The *binary matrix for set system* $\mathcal{F}$ is conversely constructed and denoted by $M^{\mathcal{F}}$. Thus, $M^{\mathcal{F}_M} = M$.

A set system $\mathcal{F}$ from universe U, $|U| = n$ has the *consecutive-ones property* if there exists a linear order or permutation $\sigma = w_1 w_2 \dots w_n$ that can be applied to U such that each set $S \in \mathcal{F}$ becomes a consecutive subsequence[12] $w_i w_{i+1} \dots w_{i+k-1}$ on σ for some positive integer $i \leq n + 1 - k$ where $k = |S|$.

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

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

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

Definition 2.2.3 (Orthogonal sets [Nov89, MM96, McC04][13]). Let $\mathcal{F}$ be a set system with universe U and sets $A, B \in \mathcal{F}$.

1. A and B are said to have a *trivial intersection* if $A \cap B$ is one of the following.
   i. $\emptyset$
   ii. A
   iii. B[14]
2. A and B are called *mutually orthogonal* or A is *orthogonal to* B and vice versa, if they have a trivial intersection.
3. *Trivial subsets* of a universe U, denoted by $\mathcal{T}(U)$, are sets that have trivial intersections with any set in 2^U. These sets are U, singleton sets in 2^U and $\emptyset$ [Nov89, MM96]. Thus, $\mathcal{T}(U) = \{U\} \bigcup \{\{v\} \mid v \in U\} \bigcup \{\emptyset\}$.
4. *Orthogonal sets of a set system* $\mathcal{F}$ with universe U are subsets of U that are orthogonal to all sets in $\mathcal{F}$. The set of all orthogonal sets to $\mathcal{F}$ is denoted by

[12] Also termed an *interval*

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

[14] In other words, A and B are either disjoint or one is the subset of the other.

$\mathcal{F}^{\perp}$[15].

5. $\mathcal{F}$ is called *complete*[16] if the following hold true for every pair of non-orthogonal[17] sets A, B in $\mathcal{F}$.

   i. $\mathcal{T}(U) \subset \mathcal{F}$

   ii. $A \cup B \in \mathcal{F}$

   iii. $A \cap B \in \mathcal{F}$

   iv. $A \setminus B \in \mathcal{F}$

   v. $B \setminus A \in \mathcal{F}$

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

6. $\overline{\mathcal{F}}$ represents the smallest super set system of $\mathcal{F}$ that is complete[18]

Generalized PQ-tree or gPQ-tree is a data structure defined in [Nov89] to represent all orthogonal sets of a set system $\mathcal{F}$. A data structure with the same name was later defined in [McC04] as part of a *substitution decomposition* for a set system $\mathcal{F}$ and subsequently [McC04] gives a new characterization of $\mathcal{F}$ using a so-called incompatibility graph. PQR-tree is defined in [MM96] as a data structure to represent any set system $\mathcal{F}$ with additional information in their R-nodes if $\mathcal{F}$ has no COP. All three of these data structures are proposed as generalizations of [BL76]'s PQ-tree and hence produce the PQ-tree if $\mathcal{F}$ has COP. As a whole, all these three data strutures are largely identical with their differences being notional. In this section, we will discuss the basic theory that they all hold. We will predominantly use the terminology from [MM96] and refer to the data structure as PQR-tree.

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

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

1. The intersection $A \cap B$

[15] [McC04, Def. 3.1] uses the term *non-overlapping family* of $\mathcal{F}$ and denotes it by $\mathcal{N}(\mathcal{F})$.
[16] [McC04] calls this a *weakly partative family*.
[17] Or overlapping.
[18] [McC04, Def. 3.2] calls this the *weak closure* of $\mathcal{F}$ denoted by $\mathcal{W}(\mathcal{F})$.

2. The union $A \cup B$ if $A \cap B \neq \emptyset$

3. The relative complements $A \setminus B$ and $B \setminus A$ if $B \nsubseteq A$ and $A \nsubseteq B$ respectively.

4. Also note that trivially, sets in $\mathcal{T}(U)$ are consecutive in any permutation of U[19].

Theorem 2.2.4 ([MM96, Th. 3,6]). *For any set system $\mathcal{F}$ we have the following.*

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

Proposition 2.2.5. Any set that is consecutive on all the COP permutations of $\mathcal{F}$ is present in $\overline{\mathcal{F}}$.

Proposition 2.2.5 is owing to the following theorem by which [MM96] describes a way to decompose the problem of finding all COP orders of $\mathcal{F}$ into two subproblems using sets in $\overline{\mathcal{F}} \cap \mathcal{F}^{\perp}$.

Theorem 2.2.6 ([MM96, Th. 7]). *For any set system $\mathcal{F}$, and $\emptyset \neq H \in \overline{\mathcal{F}} \cap \mathcal{F}^{\perp}$ we have the following.*

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

The idea behind Theorem 2.2.6 is as follows. A permutation α of U is a composition of two permutations with respect to H - i) a permutation γ of H and (ii) a permutation β of U/H.

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

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

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

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

[19] $\emptyset$ is considered consecutive by convention.

We observe that this idea of decomposing the COP problem into two subproblems in [MM96] is very similar to the substitution decomposition of a set system given in [McC04, Sec. 4].[20]

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

Corollary 2.2.7 ([MM96, Cor. 8]). *Let $\mathcal{F}$ be a set system with universe U and H is a non-empty orthogonal set $H \in \overline{\mathcal{F}} \cap \mathcal{F}^{\perp}$. If there is a PQR-tree T_1 that encodes all permutations in $\texttt{valid}\,(U/H, \mathcal{F}/H)$ and also a PQR-tree T_2' that encodes all permutations $\texttt{valid}\left(H, \mathcal{F} \cap 2^H\right)$, then a PQR-tree T for $\mathcal{F}$ can be obtained by replacing the leaf h in T_1 by T_2.*

Thus we have a recursive algorithm that can compute the PQR-tree for $\mathcal{F}$ provided we find an element from $\overline{\mathcal{F}} \cap \mathcal{F}^{\perp}$ in each iteration. The non-empty sets in $\overline{\mathcal{F}} \cap \mathcal{F}^{\perp}$ are called *node sets* since they form the nodes in the PQR-tree. They are calculated as follows. One is by computing the overlap components of $\mathcal{F}$. The overlap components is the partition that results from the overlap equivalence relation which is nothing but non-orthogonal equivalence relation[21]. Overlap components are linearly computable[MM95, Hsu92]. Once these elements are factored out, the rest of the node sets are obtained by identifying *twin* elements[22]. Two elements $a, b \in U$ are twins if their membership in every set of $\mathcal{F}$ is in tandem with each other, i. e. $\{a, b\} \perp \mathcal{F}$. This is clearly an equivalence relation and their equivalence classes is known to be computable in linear time[Hsu01, ?] and even in logspace[KKLV10].

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

Theorem 2.2.8 ([MM96, Th. 9],[McC04, Th. 2.1]). *If $\mathcal{F}$ is a set system with universe U and $|U| \geq 3$ then one of the following statements hold.*

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

[20] Using the theory cited in footnotes 15,16,18.
[21] It is easy to verify that this relation is indeed an equivalence relation.
[22] [KKLV10, Sec. 3] calls this indistinguishable elements. The equivalence class is called a *slot*.

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

Theorem 2.2.8 and the theory leading to it is very similar to [McC04, Th. 2.1, 3.5. Also Th. 3.2, 3.3, 3.4] which categorizes the nodes in the above the three cases as *prime*, *linear* and *degenerate* respectively. Their generalized PQ-tree is created in similar ways as PQR-tree above. This tree is in essense the Hasse diagram of what they call *strong elements* of $\overline{\mathcal{F}}$. Strong elements of a set family are elements that do not overlap with any other elements in the family, i. e. it is orthogonal to all other sets[McC04, Def. 3.3].

Theorem 2.2.9 ([MM96], [McC04, Th. 3.6]). *The set system $\mathcal{F}$ has COP if and only if its PQR-tree has no R nodes.*

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

As mentioned before, we observe the three data strutures of PQR-tree, gPQ-tree and generalized PQ-tree to be equivalentand the theory of all these data structures is summarised in Table 2.3.

We will now see one more generalization of PQ-tree before seeing other approaches to solving COP testing including ICPIA which is a set theoretic characterization of COP.

2.2.4 PC-tree– a generalization of PQ-tree

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

gPQ-tree [Nov89]	PQR-tree [MM96]	generalized PQ-tree [McC04]
Trivial intersections	Trivial sets $\mathcal{T}(U)$	
Trivially intersecting sets	Mutually orthogonal sets, $A \perp B$	Non-overlapping sets
	Complete collection	Weakly partative family
	$\overline{\mathcal{F}}$	Weak closure, $\mathcal{W}(\mathcal{F})$
	$\mathcal{F}^{\perp}$	Non-overlapping family, $\mathcal{N}(\mathcal{F})$
	Sets in $\mathcal{F}$ orthogonal to all other sets in $\mathcal{F}$	Strong elements
	PQR tree def.	Decomposition tree of $\mathcal{W}(\mathcal{F})$, $T(\mathcal{W}(\mathcal{F}))$ PQR tree def.
	Node sets, $(\mathcal{F} \cap \mathcal{F}^{\perp}) \setminus \{\emptyset\}$	
	P-node $\Leftrightarrow \mathcal{F} = \mathcal{T}(\mathcal{F})$ Q-node $\Leftrightarrow \mathcal{F} = consec(\alpha)$ R-node $\Leftrightarrow \mathcal{F} = 2^U$	give def of P Q R nodes

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

unrooted is necessary in order to use PC-tree as a data structure for encoding circular ordering. Definition 2.2.4 defines PC-tree.

Definition 2.2.4. [*PC-tree* [Hsu01, Dom08].] A PC-tree of matrix M with CROP on columns (rows), is a tree with the following properties.

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

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

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

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

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

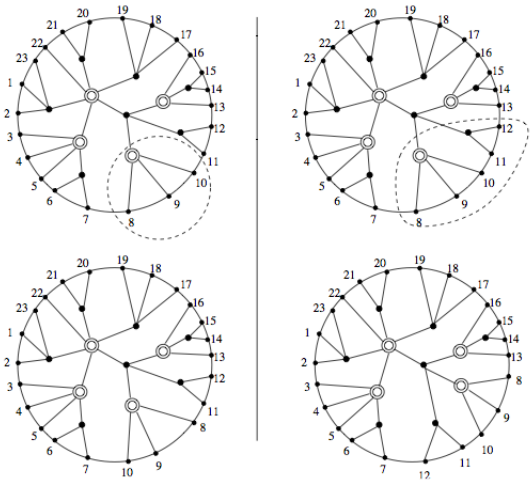


Figure 2: The PC tree can be viewed as a gadget for generating the circular-ones permutations of the columns. The C nodes are represented by double circles and the P nodes are represented by black dots. The subtree lying at one side of an edge can be flipped over to reverse the order of its leaves. The order of leaves of a consecutive set of subtrees that would result from the removal of a P node can also be reversed. All circular-ones arrangements can be obtained by a sequence of such reversals.

c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
1	**1**	0	0	0	0	0	0	0	0
0	**1**	**1**	0	0	0	0	0	0	0
1	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
0	0	0	**1**	**1**	0	0	0	0	0
0	0	0	0	**1**	**1**	0	0	0	0
0	0	0	0	0	0	**1**	**1**	0	0
0	0	0	0	0	0	0	**1**	**1**	0
0	0	0	0	0	0	0	0	0	**1**

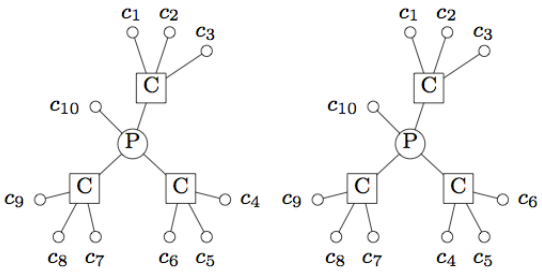


Figure 2.5: PC-tree PLACEHOLDER IMGS [HM03, Dom08]

As a data structure when PC-tree is compared with PQ-tree, the differences are, (i) it is unrooted, (ii) it represents all CROP order of a matrix (iii) it has C nodes instead of Q nodes which can be "mirrored" (operation defined in Definition 2.2.4 v). The algorithms of construction of PQ-tree in [BL76] and that of PC-tree in [Hsu01, HM03] starkly differ since the latter is a much simplified procedure.

2.2.5 ICPIA - a set cardinality based COP test

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

Definition 2.2.5 (Intersection Cardinality Preserving Interval Assignment (ICPIA)). Let $\mathcal{F} = \{A_i \mid A_i \subseteq U, i \in [m]\}$ be a set system from universe U and the set of ordered pairs $\Pi = \{(A_i, B_i) \mid B_i$ is an interval from $[n], i \in [m]\}$ be an *interval assigment* of $\mathcal{F}$, then it is called an *ICPIA* if it has the following properties.

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

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

Theorem 2.2.10 ([NS09, Th. 1]). *If an interval assignment* Π *is feasible, then it is an ICPIA.*

The necessity of ICPIA for a set system to have COP given in Theorem **??** is fairly obvious. It turns out that it is sufficient too. This is demostrated by two algorithms which work as follows. The first one iterates over the assignment Π and changes it in each iteration till the altered assignment has no more overlapping intervals[23] [NS09, Alg. 1]. This is acheived by replacing two overlapping assignment pairs (A_i, B_i) and (A_j, B_j) with the partitions of $(A_i \cup A_j, B_i \cup B_j)$ induced by their overlaps – $(A_i \setminus A_j, B_i \setminus B_j)$, $(A_i \cap A_j, B_i \cap B_j)$ and $(A_j \setminus A_i, B_j \setminus B_i)$.

To see it in terms of the theory explained in Section 2.2.3, this results in the weak closure of $\mathcal{F}$. Let $\mathcal{F}, \mathcal{I}$ be the set system and interval system[24], respectively, involved in the assignment Π and $\mathcal{F}', \mathcal{I}'$ be the same for the output of the algorithm Π'. It can be observed that $\mathcal{F}' \cup \mathcal{F}$ is $\overline{\mathcal{F}}$ or the weak closure of $\mathcal{F}$. The analog is

[23] Intervals are the second element in each ordered pair of Π. Def. 2.2.5

[24] A set system where all the sets are intervals.

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

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

Theorem 2.2.11 ([NS09, Th. 2]). *Let $\mathcal{F} = \{A_i \mid A_i \subseteq U, i \in [m]\}$ be a set system from universe U, $|U| = n$ and $\Pi = \{(A_i, B_i) \mid B_i$ is an interval from $[n], i \in [m]\}$ be an ICPIA. Then there exists a permutation $\sigma : [n] \to [n]$ such that $\sigma(A_i) = B_i$.*

What we saw so far in this section is checking for the feasibility of a given interval assignment. ICPIA can also be used to find an interval assignment when only the set system $\mathcal{F}$ is given. Part of the approach is similar to the overlap component idea in [FG65, Hsu02]. Once the overlap components are factored out, they can be independently assigned a subinterval from to which the sets in the component need to be mapped. This is simple because the overlap components are disjoint from each other. Now the subproblem is finding COP order for each overlap component. In each component, the set whose intersections with the rest of the sets in that component forms a single inclusion chain is chosen first and assigned the leftmost[25] interval. The next candidate set is chosen from this set's overlapping sets. The interval assigned is calculated by simple intersection cardinality means[26]. Once the next set is assigned an interval, the current assigned sets are all checked for ICPIA. This is a backtracking algorithm – if at any point ICPIA fails, another overlapping set is chosen. If all overlapping set attempts fail

[25]By convention. Rightmost can also be chosen analogously to get a different COP order.

[26]Assume that the next set overlaps on one of the "ends" of the current set and calculate the interval by shifting left or right of the current interval and adjusting its size to match the next set's cardinality.

ICPIA, a different "first set" is chosen. This is a remarkably simple algorithm due to its obvious simplicity in implementation.

2.2.6 Other COP testing algorithms

There are more COP test algorithms in the literature. We make a mention of one of them here. A simple linear time algorithm was presented by [HMPV00]. They use something called *Lex-BFS ordering* of the vertices of a graph to decide in linear time whether the graph is an interval graph. They also show how any interval graph recognition algorithm can be used to recognize matrices with COP. This is shown in Theorem 2.2.12.

Theorem 2.2.12 ([HMPV00, Th. 2]). *For a binary matrix M the following statements are equivalent.*

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

2.3 Optimization problems in COP

COP is a very beneficial property since it simplifies the structure of the input[27] leading to efficient algorithms to otherwise hard problems. While Section 2.2 discusses matrices with COP and how to compute COP orders, this section describes the problems that are of interest when a matrix does not have this property. The central questions in this area are (i) how close is the matrix to having COP, mainly in terms of Tucker's forbidden submatrices, and (ii) how optimally can one alter the matrix to attain COP. With regard to the latter question, recent literature indicate that there has been a lot of interest in matrix modification problems to make a matrix have COP [HG02, TZ07]. However, to tackle this problem it is more than insightful to solve question (i).

TEXT INTRODUCING SECTION

[27] Binary matrix or set system.

2.3.1 Incompatibility graph - a certificate for no COP

When a COP test algorithm reports negative, it is hard to verify this. While [Tuc72] gives forbidden substructures for COP, there are no efficient algorithms to find these strutures in a matrix. In fact, [Vel82] shows that even finding a forbidden submatrix with at least k rows in an input matrix is NP-complete. This was proven by polynomially reducing the longest path problem for graphs to this problem.

[McC04] describes another characterization for matrices without COP which is efficiently computable. This is done by way of an *incompatibility graph* of a set system[28]. The construction of this graph is based on the following observation.

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

For the sake of argument, suppose there exists such a COP order. Clearly, it will not succeed in mapping S to an interval since $b \notin S$ which is a contradiction to the assumption that this is a COP order. The ordered pair (x, y) denotes that in the COP order y comes after x. Based on above observations, this means (a, b) and (b, c) are not compatible in any COP order. So are (c, b) and (b, a). Thus a binary relation *is incompatible with* on $U \times U \setminus \{(a, a) \mid a \in U\}$ can be defined based on the membership of elements of U in the sets in $\mathcal{F}$. [McC04] creates an incompatiblity graph with vertices denoting ordered pairs and edges denoting the incompatibility relation.

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

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

[28] Since COP in matrices and set systems are equivalent problems as seen in Section 2.2.3 we will use the term set system here.

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

Theorem 2.3.1 ([McC04, Th. 6.1], [Dom08, Corrected by Th. 2.6, Proof p. 44–47]). *Let $\mathcal{F}$ be a set family with universe U. Then $\mathcal{F}$ has COP if and only if its incompatibility graph is bipartite and if it does not have the COP, the incompatibility graph has an odd cycle of length at most $n + 3$.*

Each edge of the cycle is labeled with a set from $\mathcal{F}$ that documents the incompatibility.

how does it give clues about the R node? See dom.

REST OF THE SUBSECTIONS

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.

couple of lines refering to tucker's submatrices. refer earlier section.

Once a matrix has been detected to not have COP (using any of the COT algorithms mentioned earlier), it is naturally of interest

1. 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.

2. $(*, \Delta)$-*matrices* are matrices with no restriction on number of **1**s in any column but has at most Δ **1**s in any row. Min COS-R (Min COS-C), Max COS-R (Max COS-C) are similar problems which deals with *inducing COP* on a matrix.

   (a) 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.

   (b) 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.

   Given a matrix M with no COP, [Boo75] shows that finding a submatrix M' with all columns but a maximum cardinality subset of rows such that M' has COP is NP complete. [HG02] corrects an error of the abridged proof of this reduction as given in [GJ79]. [Dom08] discusses all these problems in detail giving an extensive survey of the previously existing results which are almost exhaustively all approximation results and hardness results. Taking this further, [Dom08] presents new results in the area of parameterized algorithms for this problem.

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

4. [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.

2.4 COP in Graph Isomorphism

The survey from kklv10 conclusion.

CHAPTER 3

Tree Path Labeling of Path Hypergraphs - New Results

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

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 Summary of Proposed Problems

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.

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 $\ell : \mathcal{F} \to \mathcal{P}$, such that Feasible Tree Path Labeling returns **true** on $(\mathcal{F}, T, \ell)$.

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

Feasible Tree Path Labeling

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

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

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

3.2 Preliminaries to new results

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

3.2.1 Hypergraph Preliminaries

Definition 3.2.1 (Set systems and hypergraphs). 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* 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 $\mathcal{F}$.

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 if and only if 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 $\mathcal{F}' \cong \mathcal{F}''$, if and only if there exists a bijection $\phi : supp(\mathcal{F}') \to supp(\mathcal{F}'')$ such that for all sets $A \subseteq supp(\mathcal{F}')$, A is a hyperedge in $\mathcal{F}'$ if and only if B is a hyperedge in $\mathcal{F}''$ where $B = \{\phi(x) \mid x \in A\}$ [KKLV10], written as $B = \phi(A)$. This is called *hypergraph isomorphism*.

Definition 3.2.2 (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.

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 $\mathcal{F}$.[1]

To illustrate further, let $g : V(\mathcal{F}) \rightarrow V(\mathcal{P})$ be the above mentioned isomorphism where $V(\mathcal{F})$ and $V(\mathcal{P})$ are the vertex sets that represent the hyperedges for each hypergraph respectively, $V(\mathcal{F}) = \{v_S \mid S \in \mathcal{F}\}$ and $V(\mathcal{P}) = \{v_P \mid P \in \mathcal{P}\}$. Then the path labeling 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.

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 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 $\ell' : V(G) \rightarrow \mathcal{P}$. Moreover, for the purposes of this paper, we require that in a path labeling, $supp(\mathcal{P}) = V(T)$. If 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 $\ell_\phi = \ell$. 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 $\ell_\phi = \ell$.

Definition 3.2.3 (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 if and only if 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; $S \between S'$ if and only if$S \cap S' \neq \emptyset, S \nsubseteq S', S' \nsubseteq S$. Thus $\mathbb{O}(\mathcal{F})$ is a spanning subgraph of

[1]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 $\mathcal{P}$.

$\mathbb{I}(\mathcal{F})$ 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*.

Definition 3.2.4 (k-subdivided star). 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 —

Definition 3.2.5 (Partially ordered sets). 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 the feasibility of a 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, $|S_1 \cap S_2| - |S_1 \cap S_2 \cap S_3| = |P_1 \cap P_2| - |P_1 \cap P_2 \cap P_3| = |(P_1 \cap P_2) \setminus P_3| = |P_1 \cap (P_2 \setminus P_3)|$. □

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 path hypergraph isomorphism function. Algorithm 3 recursively does two levels of filtering of $(\mathcal{F}, \ell)$ to make it simpler in terms of set intersections while retaining the set of isomorphisms, if any, between $\mathcal{F}$ and $\mathcal{F}^\ell$. First, we present Algorithm 1 which describes `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.

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

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 $(\mathcal{F}', \ell')$

to be feasible. Assume the lemma is true till $j-1$th iteration. i. e. every hypergraph isomorphism $\phi : supp(\mathcal{F}_{j-1}) \to V(T)$ that defines $(\mathcal{F}, \ell)$'s feasibility, is such that the induced path labeling on $\mathcal{F}_{j-1}$, say denoted by $\ell_{\phi[\mathcal{F}_{j-1}]}$, 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 $\mathcal{F}_{j-1}$ adding or removing no elements. For the same reason and ϕ being a bijection, it is clear that when applying the ϕ-induced path labeling on $\mathcal{F}_j$, $\ell_{\phi[\mathcal{F}_j]}(S_1 \setminus S_2) = \ell_{\phi[\mathcal{F}_{j-1}]}(S_1) \setminus \ell_{\phi[\mathcal{F}_{j-1}]}(S_2)$. 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 $\ell_{\phi[\mathcal{F}_j]} = \ell_j$. □

Lemma 3.3.3. *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 $R = S_1 \cap S_2$, $|R| = |S_1 \cap S_2| = |\ell_{j-1}(S_1) \cap \ell_{j-1}(S_2)| = |\ell_j(S_1 \cap S_2)| = |\ell_j(R)|$. If $R = S_1 \setminus S_2$, $|R| = |S_1 \setminus S_2| = |S_1| - |S_1 \cap S_2| = |\ell_{j-1}(S_1)| - |\ell_{j-1}(S_1) \cap \ell_{j-1}(S_2)| = |\ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2)| = |\ell_j(S_1 \setminus S_2)| = |\ell_j(R)|$. Similarly if $R = S_2 \setminus S_1$.

✠ *Invariant* III

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

IIIb | *Only one, say R, is a new set.* Due to invariant IV induction hypothesis, Lemma 3.3.1 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 3.3.4. *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 3.3.2 $(\mathcal{F}_j, \ell_j)$ for every iteration $j > 0$ is feasible. Also, every hypergraph isomorphism $\phi : supp(\mathcal{F}) \to V(T)$ that induces 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 if and only if 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 3.3.3 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 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.

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

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** $(\mathcal{F}', \ell')$

hypergraph isomorphism. The following lemma shows that this will not happen.

Lemma 3.3.5. *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 3.3.6. *In Algorithm 2, for all $j > 0$, at the end of the jth iteration of the* **while** *loop the four invariants given in Lemma 3.3.3 hold.*

Proof. By Lemma 3.3.5 we know that set X will not be empty in any iteration of the **while** loop if input ICPPL $(\mathcal{F}, \ell)$ is feasible and 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 3.3.5, 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 for `filter_1` and Algorithm 2 for `filter_2` which when executed in that order result in a new ICPPL on the same universe U and target tree T. We also proved that if the input is indeed feasible, these algorithms executed in sequence give an ICPPL that preserves a subset of hypergraph isomorphisms as that of the input ICPPL. Now we present the algorithmic characterization of a feasible tree path labeling by way of Algorithm 3.

Prove that the same set of hypergraph isomorphisms are in the filtered ICPPL? – NOPE. we are only interested in testing feasibility. not necessarily giving ALL bijections.

Lemma 3.3.7. *PROOF TBD. Let $(\mathcal{F}_2, \ell_2)$ be the output of* `filter_2` *and the ICPPL input to* `filter_1` *preceeding its call is $(\mathcal{F}, \ell)$ with target tree T. If a bijection $\phi : supp(\mathcal{F}_2) \to V(T)$ is a hypergraph isomorphism of $(\mathcal{F}_2, \ell_2)$, then it is a hypergraph isomorphism of $(\mathcal{F}, \ell)$.*

Algorithm 3 computes a hypergraph isomorphism ϕ recursively using Algorithm 1 and Algorithm 2 and pruning the leaves of the input tree. In brief, it is done as follows. If the input is feasible, the output of `filter_1`, say $(\mathcal{F}_1, \ell_1)$, gives a new ICPPL which preserves all the original hypergraph isomorphisms. This ICPPL $(\mathcal{F}_1, \ell_1)$ is then given to `filter_2` which gives the second filtered ICPPL, say $(\mathcal{F}_2, \ell_2)$ such that a feasible leaf labeling is in singleton elements of $\mathcal{F}_2$. These leaf labels are the elements in $supp(\mathcal{F})$ that map to leaves in T in at least one hypergraph isomorphism of the input ICPPL ℓ. It must be noted that at this point, the algorithm "chooses" one (or more - there could be more than one bijections that have the same leaf labels) hypergraph isomorphism ϕ', from the set of all isomorphisms resulting from the feasibility of the original input $(\mathcal{F}, \ell)$. However, if there exists one (i. e. input is feasible), it will be found – thus performing the test required in Feasible Tree Path Labeling.

Since their pre-images have been found, all leaves in T are then pruned away. The leaf labels are removed from the path labeling ℓ_2 and the corresponding elements are removed from the corresponding sets in $\mathcal{F}_2$. It is now clear that we have

a subproblem with a new hypergraph $\mathcal{F}'$, new tree path labeling ℓ' and target tree T'. The tree pruning algorithm is recursively called on $\mathcal{F}', \ell', T'$. The recursive call returns the bijection ϕ'' for the rest of the elements in $supp(\mathcal{F})$ which along with the leaf labels ϕ' computed earlier gives us a hypergraph isomorphism ϕ for the input $\mathcal{F}, \ell, T$. The following lemma formalizes the characterization of feasible path labeling.

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` $\phi' : supp(\mathcal{F}) \to L$
  Remove $\{x\}$ and $\{v\}$ from $\mathcal{F}'$, ℓ' appropriately
end for
$T' \leftarrow T \setminus L$
$\phi'' \leftarrow$ `get-hypergraph-isomorphism` $(\mathcal{F}', \ell', T')$
$\phi \leftarrow \phi'' \cup \phi'$
return ϕ

Lemma 3.3.8. *If $(\mathcal{F}, \ell)$ is an ICPPL from a tree T and Algorithm 3, on input $\mathcal{F}, \ell, T$ returns a non-empty function, then there exists a hypergraph isomorphism $\phi : supp(\mathcal{F}) \to V(T)$ 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 input target tree T (of the current recursive call). Moreover, by Lemma 3.3.4 and Lemma 3.3.5 it is consistent with the input tree path labeling ℓ (of the current recursive call). The tree pruning is done by only removing leaves in each call to the function and is done till the tree becomes empty. Thus the returned function $\phi : supp(\mathcal{F}) \to V(T)$ is a union of mutually exclusive one-to-one functions exhausting all vertices of the tree. In other words, it is a bijection from $supp(\mathcal{F})$ to $V(T)$ inducing the given path labeling ℓ and thus a hypergraph isomorphism. □

Theorem 3.3.9. *A path labeling $(\mathcal{F}, \ell)$ on tree T is feasible if and only if it is an ICPPL and Algorithm 3 with $(\mathcal{F}, \ell, T)$ as input returns a non-empty function.*

Proof. From Lemma 3.3.8, we know that if $(\mathcal{F}, \ell)$ is an ICPPL and if Algorithm 3 with $(\mathcal{F}, \ell, T)$ as input returns a non-empty function, then $(\mathcal{F}, \ell)$ is feasible. Now consider the case where $(\mathcal{F}, \ell)$ is feasible, i. e. there exists a hypergraph isomorphism ϕ such that $\ell_\phi = \ell$. Lemma 3.3.4 and Lemma 3.3.5 show us that `filter_1` and `filter_2` do not exit if input is feasible. Thus Algorithm 3 returns a non-empty function. □

3.4 Computing feasible TPL with special target trees

Section 3.3 described properties that a TPL must have for it to be feasible. The next problem of interest is to test if a given hypergraph is a path hypergraph with respect to a given target tree[2]. 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[3], has been solved using a polynomial time algorithm. The latter problem enforces some conditions on the hypergraph too which will be seen in Section 3.4.2.

3.4.1 Target tree is a Path

Let us consider a special case of ICPPL with the following properties. The target 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.

[2]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.

[3]See Section 3.2 for the formal definition.

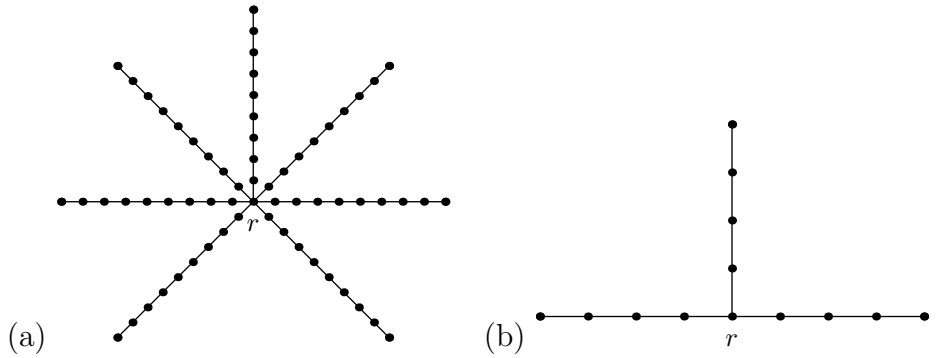


Figure 3.1: (a) 8-subdivided star with 7 rays (b) 3-subdivided star with 3 rays

3.4.2 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 $\langle v_{i1}, \ldots, v_{i(k+2)} = r \rangle$ 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 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.

repetition

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

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

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

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

Proof of Correctness and Analysis of Running Time: It is clear that the algorithm runs in polynomial time, as at each step, at most three-way intersection cardinalities need to be checked. Further, finding super-marginal hyperedges can also be done in polynomial time, as it involves considering the overlap regions and checking if the inclusion partial order contains a single minimal element. In particular, once the super-marginal edges are identified, each iteration 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 $X \nsubseteq Y_1$ such that $Y_1 \between X$ and Y_1 has at least one vertex closer to the leaf in R_i than any vertex in X. Similarly with the same argument there exists $Y_2 \in \mathcal{F}$ with same subset and overlap relation with X except it has has at least one vertex farther away from the leaf in R_i than any vertex in X. Clearly $Y_1 \cap X$ and $Y_2 \cap X$ cannot be part of same inclusion chain which contradicts that assumption X is super-marginal. Thus the claim is proved. □ □

Lemma 3.4.2. *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. `abstract end`

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.

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

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

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

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, $M_{ij} = 1$ if and only if $x_i \in S_j$. 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 if and only if 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 $E_f = \{(v_i, v_j) \mid S_i \cap S_j \neq \emptyset$ and $S_i \nsubseteq S_j, S_j \nsubseteq S_i\}$. We use this graph to decompose M as described in [Hsu02, NS09]. A prime sub-matrix of M is defined as the matrix formed by a set of columns of M which correspond to a connected component of the graph G_f. Let us denote the prime sub-matrices by $M_1, \ldots, M_p$ each corresponding to one of the p components of G_f. Clearly, two distinct matrices have a distinct set of columns. Let $col(M_i)$ be the set of columns in the sub-matrix M_i. The support of a prime sub-matrix M_i is defined as $supp(M_i) = \bigcup_{j \in col(M_i)} S_j$. 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 if and only if 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 if and only if the matrix has COP. Indeed, this is an algorithmic characterization of the consecutive ones property and the absence of the PQ-tree does not yield any witness to the reason for failure. A closely related data structure is the generalized PQ tree in [McC04]. In generalized PQ tree the P and Q nodes are called prime and linear nodes. Aside from that, it has a third type of node

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

Theorem 3.6.1 (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.

CHAPTER 4

Conclusion

MOVED FROM INTRO SUBSECTION IN ch:myresearch CHAPTER begin

It is an interesting fact that for a matrix with the COP, the intersection graph of the corresponding set system is an interval graph. A similar connection to a subclass of chordal graphs and a superclass of interval graphs exists for the generalization of COP. In this case, the intersection graph of the corresponding set system must be a *path graph*. Chordal graphs are of great significance, are extensively studied, and have several applications. One of the well known and interesting properties of a chordal graphs is its connection with intersection graphs [Gol04]. For every chordal graph, there exists a tree and a family of subtrees of this tree such that the intersection graph of this family is isomorphic to the chordal graph [Ren70, Gav78, BP92]. These trees when in a certain format, are called *clique trees* [PPY94] of the chordal graph. This is a compact representation of the chordal graph. There has also been work done on the generalization of clique trees to clique hypergraphs [KM02]. If the chordal graph can be represented as the intersection graph of paths in a tree, then the graph is called path graph [Gol04]. Therefore, it is clear that if there is a bijection from U to $V(T)$ such that for every set, the elements in it map to vertices of a unique path in T, then the intersection graph of the set system is indeed a path graph. However, this is only a necessary condition and can be checked efficiently because path graph recognition is polynomial time solvable[Gav78, Sch93]. Indeed, it is possible to construct a

set system and tree, such that the intersection graph is a path graph, but there is no bijection between U and $V(T)$ such that the sets map to paths. Path graph isomorphism is known be isomorphism-complete, see for example [KKLV10]. An interesting area of research would be to see what this result tells us about the complexity of the tree path labeling problem (not covered in this paper).

MOVED FROM INTRO SUBSECTION IN ch:myresearch CHAPTER end

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

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

While we address the computation of TPL for a given hypergraphfrom a given target tree in this research, however optimization problems in TPL, akin to problems in Section 1.4.2 for COP, is outside the scope of this thesis. Whether TPL on general trees is solvable in P remains open. So do optimization opportunities in TPL (possible extensions to optimization for COP in the Section 1.4.2.).

APPENDIX A

More proofs

Lemma A.0.4. *Consider four paths in a tree Q_1, Q_2, Q_3, Q_4 such that they have nonempty pairwise intersection and Q_1, Q_2 share a leaf. Then there exists distinct $i, j, k \in \{1, 2, 3, 4\}$ such that, $Q_1 \cap Q_2 \cap Q_3 \cap Q_4 = Q_i \cap Q_j \cap Q_k$.*

Proof. Case 1: w.l.o.g, consider $Q_3 \cap Q_4$ and let us call it Q. This is clearly a path (intersection of two paths is a path). Suppose Q does not intersect with $Q_1 \setminus Q_2$, i.e. $Q \cap (Q_1 \setminus Q_2) = \emptyset$. Then $Q \cap Q_1 \cap Q_2 = Q \cap Q_2$. Similarly, if $Q \cap (Q_2 \setminus Q_1) = \emptyset$, $Q \cap Q_1 \cap Q_2 = Q \cap Q_1$. Thus it is clear that if the intersection of any two paths does not intersect with any of the set differences of the remaining two paths, the claim in the lemma is true. *Case 2:* Let us consider the compliment of the previous case. i.e. the intersection of any two paths intersects with both the set differences of the other two. First let us consider $Q \cap (Q_1 \setminus Q_2) \neq \emptyset$ and $Q \cap (Q_1 \setminus Q_2) \neq \emptyset$, 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 □

BIBLIOGRAPHY

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

[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 fine 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.

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

[Gav74] Fanica Gavril. Algorithms on circular-arc graphs. *Networks*, 4:357369, 1974.

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

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

[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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[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.

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

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

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

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

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

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

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