*SYNOPSIS OF*

# Generalization of Consecutive Ones Property of Binary Matrices

*A THESIS*

*submitted by*

**ANJU SRINIVASAN**

*for the award of the degree*

*of*

**MASTER OF SCIENCE**

(by Research)


**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**
**INDIAN INSTITUTE OF TECHNOLOGY MADRAS.**

**NOVEMBER 2011**

# 1 Introduction

## 1.1 Illustration of the problem

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

Let $U$ and $\mathcal{F}$ represent the set of students and the set of study groups respectively and the integers $n$ and $m$ denote the total number students and study groups respectively. In relation to this example, these are defined in Table 1. Also given there is the study group allocation to students.

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

Table 1: Students and study groups in *Wallace Studies Institute*

The campus has a residential area *Infinite Loop* that has $n$ single occupancy apartments reserved for the study groups' accommodation. All these apartments are located such that the streets connecting them do *not* form loops. Fig 1.1 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 enforcing additional conditions, namely, that all the

students in a study group must be next to each other; in other words, for one student to reach another fellow study group member's apartment (for all study groups the student is part of), she must not have to pass the apartment of any student who is not in that study group. To further elucidate, the apartments of students of any study group must be arranged in an exclusive unfragmented path on the street map. Exclusivity here means that the path must not have apartments from other study groups (unless that apartment is also part of *this* study group).

An intuitive approach to this problem would be to first find the paths that each study group decides to inhabit and then refine the allocation to individual students. A feasible allocation of exclusive routes to study groups is illustrated in Fig **??** and the students' allocation of apartments that obeys this route allocation is also shown. How this is algorithmically computed is the focus of this thesis.

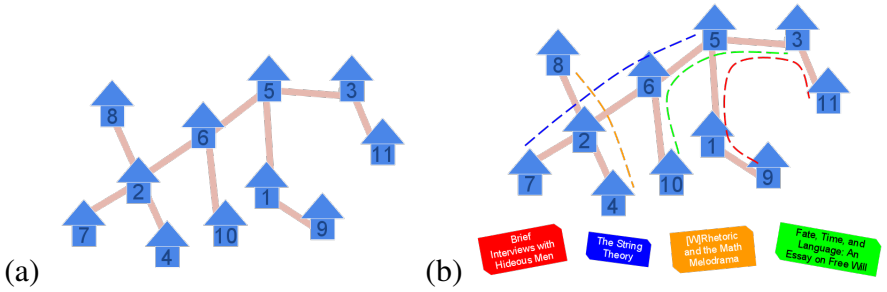


Figure 1: (a) *Infinite Loop* street map (b) *Infinite Loop* street map with study group routes allocated. Routes are color coded as follows: red for $\mathbb{B}$ group, blue for $\mathbb{T}$ group, orange for $\mathbb{W}$ group, green for $\mathbb{F}$ group

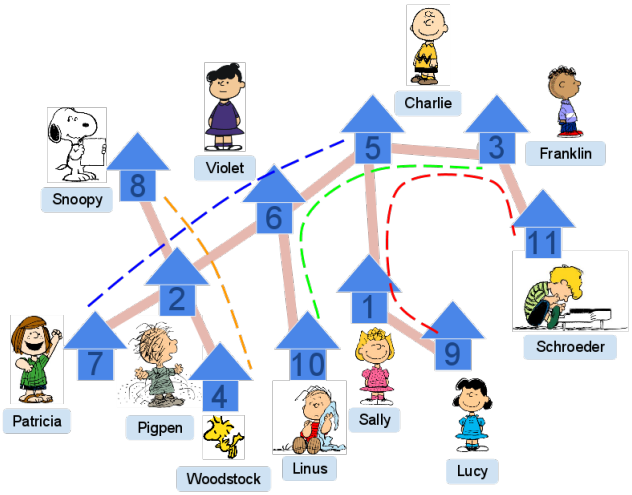


Figure 2: Individual allocation of apartments to students in *Infinite Loop* that meets the requirements stated before.

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

The interval assignment problem to a set system is equivalent to the consecutive ones problem (COP) 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. Consecutive ones property (COP) of binary matrices 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

$M_1$:

| $c_1$ | $c_2$ | $c_3$ | $c_4$ |
|---|---|---|---|
| **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 3: Matrices with and without COP. $M_1$ has COP because by permuting its columns, $c_1$-$c_4$, one can obtain $M_1'$ where the **1**s in each row are consecutive. $M_2$, however, does not have COP since no permutation of its columns, $d_1$-$d_4$, will arrange **1**s in each row consecutively [Dom08].

the study group accommodation problem).

Section 2 gives a brief survey of COP and optimization problems related to it. Section 4 presents a summary of our results on the extention of COP namely, the tree path labeling problem.

# 2 Background

## 2.1 Matrices with COP

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

Recall that a matrix with COP is one whose rows (columns) can be rearranged so that the **1**s in every column (row) are in consecutive rows (columns). Figure 3 shows examples of this property. COP in binary matrices has several practical applications in diverse fields including scheduling [HL06], information retrieval [Kou77] and computational biology [ABH98]. Further, it is a tool in graph theory [Gol04] for interval graph recognition, characterization of Hamiltonian graphs, planarity testing [BL76] and in integer linear programming [HT02, HL06].

The obvious first questions after being introduced to the consecutive ones property

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

The construction of PQ trees in [BL76] draws on the close relationship of COP matrices 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) 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 cer-

tificate 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].

## 2.2 Optimization problems in COP

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

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

Once a matrix has been detected to not have COP (using any of the COT algorithms mentioned earlier), it is naturally of interest to find out the smallest (in terms of number of rows and/or columns and/or number of entries that are **1**s) forbidden substructure. [Dom08] discusses a couple of algorithms which are efficient if the number of ones in a row is small. This is of significance in the case of sparce matrices where this

number is much lesser than the number of columns. $(*, \Delta)$-*matrices* are matrices with no restriction on number of **1**s in any column but has at most $\Delta$ **1**s in any row. MIN COS-R (MIN COS-C), MAX COS-R (MAX COS-C) are similar problems which deals with inducing COP on a matrix. In MIN COS-R (MIN COS-C) the question is to find the minimum number of rows (columns) that must be deleted to result in a matrix with COP. In the dual problem MAX COS-R (MAX COS-C) the search is for the maximum number of rows (columns) that induces a submatrix with COP. Given a matrix $M$ with no COP, [Boo75] shows that finding a submatrix $M'$ with all columns but a maximum cardinality subset of rows such that $M'$ has COP is NP complete. [HG02] corrects an error of the abridged proof of this reduction as given in [GJ79]. [Dom08] discusses all these problems in detail giving an extensive survey of the previously existing results which are almost exhaustively all approximation results and hardness results. Taking this further, [Dom08] presents new results in the area of parameterized algorithms for this problem.

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

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

# 3 Motivation

As seen in Section 2.1, [NS09] describes a characterization of consecutive ones property based on the cardinality properties of the set representations of the columns. This result

is very relevant to this thesis because aside from it simplifying COT to a great degree, our generalization problem is motivated by their results.

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

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

# 4 Summary of results

We address the computation of TPL[1] for a given set system and tree in this research. However optimization problems have not been explored.

Given a path labeling $\ell$ to $\mathcal{F}$ from a tree $T$, we give a necessary and sufficient condition for it to be a feasible path labeling. This necessary and sufficient condition can be tested in polynomial time. The most interesting consequence is that in our constructive procedure, it is sufficient to iteratively check if three-way intersection cardinalities are preserved. In other words, in each iteration, it is sufficient to check if the intersection of any three sets is of the same cardinality as the intersection of the corresponding paths. This generalizes the well studied question of the case when the given tree $T$ is a path [Hsu02, NS09], in other words the interval assignment problem or consecutive ones problem.

Aside from checking if a given TPL is feasible, we also attempt to compute the TPL for a given set system (hypergraph) and tree, if one exists. For the latter problem, we have an algorithm for general trees which runs in exponential time. We present a polynomial time algorithm for a special class of trees called $k$-subdivided trees and when the hypergraph $\mathcal{F}$ is such that all hyperedges have at most $k + 2$ elements. In spite of this restricted case we consider, we believe that our results are of significant interest in understanding the nature of Graph Isomorphism which is polynomial time solvable in interval graphs and is hard on path graphs. The connection of TPL to graph isomorphism will be made later in the document

## 4.1 Characterization of Feasible Tree Path Labelings

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

The following lemma is useful in subsequent arguments.

[1] Tree Path Labeling

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

Table 2: ICPPL properties

**Lemma 4.1** *If $\ell$ is an ICPPL, and $S_1, S_2, S_3 \in \mathcal{F}$, then $|S_1 \cap (S_2 \setminus S_3)| = |\ell(S_1) \cap (\ell(S_2) \setminus \ell(S_3))|$.*

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 **??** returns a non-empty function. Algorithm **??** recursively does two levels of filtering of $(\mathcal{F}, \ell)$ to make it simpler while retaining the set of isomorphisms, if any, between $\mathcal{F}$ and $\mathcal{F}^{\ell}$.

First, we present Algorithm **??** called `filter_1` 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 4.2** *In Algorithm **??**, 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.*

**Lemma 4.3** *In Algorithm **??**, 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* $R, R', R'' \in \mathcal{F}_j$ |

**Lemma 4.4** *If the input ICPPL $(\mathcal{F}, \ell)$ to Algorithm **??** 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 **??** will not execute.*

As a result of Algorithm **??** 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 **??** we identify elements

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

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

**Lemma 4.5** *If the input ICPPL $(\mathcal{F}, \ell)$ to Algorithm* **??** *is feasible, then for all iterations $j > 0$ of the* **while** *loop, the* **exit** *statement in line* **??** *does not execute.*

**Lemma 4.6** *In Algorithm* **??***, for all $j > 0$, at the end of the $j$th iteration of the* **while** *loop the four invariants given in Lemma 4.3 hold.*

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

Algorithm **??** computes a hypergraph isomorphism $\phi$ recursively using Algorithm **??** and Algorithm **??** and pruning the leaves of the input tree. In brief, it is done as follows. Algorithm **??** gives us the leaf labels in $\mathcal{F}_2$, i.e., the elements in $supp(\mathcal{F})$ that map to leaves in $T$, where $(\mathcal{F}_2, \ell_2)$ is the output of Algorithm **??**. All leaves in $T$ are then pruned away. The leaf labels are removed from the path labeling $\ell_2$ and the corresponding elements are removed from the corresponding sets in $\mathcal{F}_2$. This tree pruning algorithm is

recursively called on the altered hypergraph $\mathcal{F}'$, path label $\ell'$ and tree $T'$. The recursive call returns the bijection $\phi''$ for the rest of the elements in $supp(\mathcal{F})$ which along with the leaf labels $\phi'$ gives us the hypergraph isomorphism $\phi$. The following lemma formalizes the characeterization of feasible path labeling.

**Lemma 4.7** *If $(\mathcal{F}, \ell)$ is an ICPPL from a tree $T$ and Algorithm* **??**, `get- hypergraph-isomorphism` $(\mathcal{F}, \ell, T)$ *returns a non-empty function, then there exists a hypergraph isomorphism $\phi : supp(\mathcal{F}) \to V(T)$ such that the $\phi$-induced tree path labeling is equal to $\ell$ or $\ell_\phi = \ell$.*

**Theorem 4.8** *A path labeling $(\mathcal{F}, \ell)$ on tree $T$ is feasible if and only if it is an ICPPL and Algorithm* **??** *with $(\mathcal{F}, \ell, T)$ as input returns a non-empty function.*

### ICPPL when given tree is a path

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

## 4.2 Case of a Special Tree – The $k$-subdivided star

In this section we consider the problem of assigning paths from a $k$-subdivided star $T$ to a given set system $\mathcal{F}$ 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 $O_1$ and $O_2$, are such that any two sets $S_1 \in O_1$ and $S_2 \in O_2$ are disjoint, or, w.l.o.g, all the

sets in $O_1$ are contained within one set in $O_2$. 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 $R_i$.

### 4.2.1 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 $O$, $T$ and vertex $r$, respectively. In particular, note that the vertices of $O$ correspond to the sets in $\mathcal{F}$, and the edges correspond to the overlap relation.

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

I. **Iteration 1:** Let $S = \{X_1, \ldots, X_s\}$ 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 $X \not\perp Y$. Further let $Z \in \mathcal{L}_1^i \cup L_2^i$ such that $Z \not\perp Y$. 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 $X \not\perp Y$, $\ell(X) \not\perp \ell(Y)$ 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.

# 5 Conclusion

Our results also have close connection to recognition of path graphs and connection to path graph isomorphism. Graphs which can be represented as the intersection graph of paths in a tree are called *path graphs*[Gol04]. Thus given a hypergraph $\mathcal{F}$, it can be viewed 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 combinatorially characterized as the intersection graphs of subtrees of a tree. Path graphs are well studied in the literature [Ren70, Gav78, BP92, Gol04]. Checking if a graph is a path graph can be done in polynomial time by the results of [Gav78, Sch93]. However, this is only a necessary condition for our question. On the other hand, path graph isomorphism is known be isomorphism-complete; see for example [KKLV10]. Therefore, it is unlikely that we can solve the problem of finding feasible path labeling $\ell$ for a given $\mathcal{F}$ and tree $T$. It is definitely interesting to classify the kinds of trees and hypergraphs for which feasible path labelings can be found efficiently. These results would form a natural generalization of COP testing and interval graph isomorphism, culminating in Graph Isomorphism itself.

As it will be described in detail later in this document, isomorphism of certain classes of graphs, namely chordal graphs, have a close relationship with consecutive

ones property and generalizations of it. This is perhaps because of how closely COP of a matrix relates to properties of graphs derived from matrices as seen in the following results. A well known result in perfect graph theory is that the maximal cliques of an interval graph $G$ can be linearly ordered such that for all $v \in V(G)$, cliques containing $v$ are consecutive in the

ordering[GH64]. This clearly means that a graph $G$ is an interval graph if and only if maximal clique vertex incidence matrix of $G$ has COP. Also, maximal cliques of any chordal graph can be enumerated in polytime $O(m + n)$ . [FG65] uses these results to give the first polynomial time algorithm for COT. A bipartite graph is convex (on $R$) if and only if its half adjacency matrix has COP on rows. The results in [BL76] on COT are based on the result that interval graphs are AT-free chordal graphs.

**Complexity challenges:** Canonization is an important tool in graph isomorphism. Invention of a deterministic method of canonization for any class of graphs naturally results in an algorithm for isomorphism. All that is required is to check if the canons of two graphs are the same. Thus complexity of graph isomorphism can be studied by studying canonization methods. While general graph ismorphism remains elusive in terms of hardness, canonization has been studied for smaller classes of graphs thus giving complexity/hardness results for them.

In 1992, [Lin92] made an important discovery that tree isomorphism is in logspace. It was done by inventing a method of canonization of trees using a logspace depth first traversal algorithm. .

[KKLV10] proved that interval graph canonization is also in logspace thus drawing logspace conslusions about COT. Interval graphs are FP+C (fixed point with counting) definable and [Lau09] showed that this implies that it captures PTIME. This result along with that of undirected graph reachability being in logspace [Rei08], [KKLV10] proved their logspace result.

# REFERENCES


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

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

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

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

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

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

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

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

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

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

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

[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.

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

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

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

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

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

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

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

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

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

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

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

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

[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.

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

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

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

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

# 6 Proposed Contents of the Thesis

The outline of the thesis is as follows:

Chapter 1 - Introduction
- Section 1.1 Consecutive Ones Testing
- Section 1.2 Matrix modification to attain COP or "almost" COP
- Section 1.3 Graph Isomorphism
- Section 1.4 Logspace complexity of graph canonization using COP
- Section 1.5 Motivation, Objective and Scope of Thesis
- Section 1.6 Summary of Results
- Section 1.7 Organization of document

Chapter 2 - Consecutive ones property
- Section 2.1 Characterization of COP
- Section 2.2 Recognition of COP
- Section 2.3 Alteration to matrices to get COP
- Section 2.4 Complexity of certain COP variations

Chapter 3 - Other problems related to COP

Chapter 4 Research
- Section 4.1 Tree path labeling of path hypergraphs
- Section 4.2 Extension of po theory from [NS09]

Chapter 5 - Conclusion

Chapter 6 - Bibliography