Tree Path Labeling of Path Hypergraphs: A Generalization of the Consecutive Ones Property

N.S. Narayanaswamy[1] and Anju Srinivasan[2,3]

Indian Institute of Technology Madras, Chennai - 600036.
[1]swamy@cse.iitm.ernet.in, [2]asz@cse.iitm.ac.in, [3]anjus.math@gmail.com

Abstract. In this paper, we explore a natural generalization of results on matrices with the Consecutive Ones Property. We consider the following constraint satisfaction problem. Given (i) a set system $\mathcal{F} \subseteq (2^U \setminus \emptyset)$ of a finite set U of cardinality n, (ii) a tree T of size n and (iii) a bijection called *tree path labeling*, ℓ mapping the sets in $\mathcal{F}$ to paths in T, does there exist at least one bijection $\phi : U \to V(T)$ such that for each $S \in \mathcal{F}$, $\{\phi(x) \mid x \in S\} = \ell(S)$? A tree path labeling of a set system is called *feasible* if there exists such a bijection ϕ. We present an algorithmic characterization of feasible tree path labeling. COP is a special instance of tree path labeling problem when T is a path. We also present an algorithm to find the tree path labeling of a given set system when T is a *k-subdivided star*.

1 Introduction

Consecutive ones property (COP) of binary matrices is a widely studied combinatorial problem. The problem is to rearrange rows (columns) of a binary matrix in such a way that every column (row) has its 1s occur consecutively. If this is possible the matrix is said to have the COP. It 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, and in integer linear programming [HT02,HL06]. Recognition of COP is polynomial time solvable by several algorithms. PQ trees [BL76], variations of PQ trees [MM96,Hsu01,Hsu02,McC04], ICPIA [NS09] are the main ones.
The problem of COP testing can be easily seen as a simple constraint satisfaction problem involving a system of sets from a universe. Every column of the binary matrix can be converted into a set of integers which are the indices of the rows with 1s in that column. When observed in this context, if the matrix has the COP, a reordering of its rows will result in sets that have only consecutive integers. In other words, the sets after reordering are intervals. Indeed the problem now becomes finding interval assignments to the given set system [NS09] with a single permutation of the universe (set of row indices) which permutes each set to its interval. The result in [NS09] characterize interval assignments to the sets which can be obtained from a single permutation of the rows. They show that for each set, the cardinality of the interval assigned to it must be same as the cardinality of the set, and the intersection cardinality of any two sets must be same as the interesction cardinality of the corresponding intervals. While this is naturally a necessary condition, [NS09] shows this is indeed sufficient. Such an interval assignment is called an Intersection Cardinality Preserving Interval Assignment (ICPIA). Finally, the idea of decomposing a given 0-1 matrix into prime matrices to check for COP is adopted from [Hsu02] to test if an ICPIA exists for a given set system.

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

set in the system maps to a path in T. We refer to this as the *tree path labeling problem* for an input set system, target tree pair - $(\mathcal{F}, T)$. As a special case if the tree T is a path, the problem becomes the interval assignment problem. We focus on the question of generalizing the notion of an ICPIA [NS09] to characterize feasible path assignments. We show that for a given set system $\mathcal{F}$, a tree T, and an assignment of paths from T to the sets, there is a feasible bijection between U and $V(T)$ if and only if all intersection cardinalities among any three sets (not necessarily distinct) is same as the intersection cardinality of the paths assigned to them and the input runs a filtering algorithm (described in this paper) without prematurely exiting. This characterization is proved constructively and it gives a natural data structure that stores all the relevant feasible bijections between U and $V(T)$. Further, the filtering algorithm is also an efficient algorithm to test if a tree path labeling to the set system is feasible. This generalizes the result in [NS09].
It is an interesting fact that for a matrix with the COP, the intersection graph of the corresponding set system is an interval graph. A similar connection to a subclass of chordal graphs and a superclass of interval graphs exists for the generalization of COP. In this case, the intersection graph of the corresponding set system must be a *path graph*. Chordal graphs are of great significance, extensively studied, and have several applications. One of the well known and interesting properties of a chordal graphs is its connection with intersection graphs [Gol04]. For every chordal graph, there exists a tree and a family of subtrees of this tree such that the intersection graph of this family is isomorphic to the chordal graph [Ren70,Gav78,BP92]. These trees when in a certain format, are called clique trees [PPY94] of the chordal graph. This is a compact representation of the chordal graph. There has also been work done on the generalization of clique trees to clique hypergraphs [KM02]. If the chordal graph can be represented as the intersection graph of paths in a tree, then the graph is called path graph [Gol04]. Therefore, it is clear that if there is a bijection from U to $V(T)$ such that for every set, the elements in it map to vertices of a unique path in T, then the intersection graph of the set system is indeed a path graph. However, this is only a necessary condition and can be checked efficiently because path graph recognition is polynomial time solvable[Gav78,Sch93]. Indeed, it is possible to construct a set system and tree, such that the intersection graph is a path graph, but there is no bijection between U and $V(T)$ such that the sets map to paths. Path graph isomorphism is known be isomorphism-complete, see for example [KKLV10]. An interesting area of research would be to see what this result tells us about the complexity of the tree path labeling problem (not covered in this paper). In the later part of this paper, we decompose our search for a bijection between U and $V(T)$ into subproblems. Each subproblem is on a set subsystem in which for each set, there is another set in the set subsystem with which the intersection is *strict*, i.e., 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 the COP in a given matrix, the COP problem is solved on an equivalent set of prime matrices.

Roadmap. The necessary preliminaries are presented in Section 2. Section 3 documents the characterization of a feasible path labeling and finally, Section 4 describes a polynomial time algorithm to find the tree path labeling of a given set system from a given k-subdivided tree.

2 Preliminaries

In this paper, 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 $\mathcal{F}$, i.e., $supp(\mathcal{F}) = \bigcup_{S \in \mathcal{F}} S$. For the purposes of this paper, a set system is required to "cover" the universe, i.e. $supp(\mathcal{F}) = U$.

The graph T represents a given tree with $|V(T)| = n$. A *path system* $\mathcal{P}$ is a set system of paths from T. Formally, $\mathcal{P} \subseteq \{P \mid P \subseteq V,\ T[P] \text{ is a path}\}$.
A set system $\mathcal{F}$ can be alternatively represented by a *hypergraph* $\mathcal{F}_H$ whose vertex set is $supp(\mathcal{F})$ and hyperedges are the sets in $\mathcal{F}$. This is a known representation for interval systems in literature [BLS99,KKLV10]. We extend this definition here to path systems. Due to the equivalence of set system and hypergraph, 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 iff their corresponding hyperedges have a non-empty intersection [Gol04].
Two hypergraphs $\mathcal{F}', \mathcal{F}''$ are said to be isomorphic (*hypergraph isomorphism*) to each other, denoted by $\mathcal{F}' \cong \mathcal{F}''$, iff there exists a bijection $\phi : supp(\mathcal{F}') \to supp(\mathcal{F}'')$ such that for all sets $A \subseteq supp(\mathcal{F}')$, A is a hyperedge in $\mathcal{F}'$ iff B is a hyperedge in $\mathcal{F}''$ where $B = \{\phi(x) \mid x \in A\}$.
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 isomorphism is $\phi : supp(\mathcal{F}) \to V(T)$, then it is clear that there is an induced path labeling $l_\phi : \mathcal{F} \to \mathcal{P}$ to the set system. Note that $supp(\mathcal{P}) = V(T)$.
[a:] Let the intersection graphs of two hypergraphs be isomorphic, $\mathbb{I}(\mathcal{F}) \cong \mathbb{I}(\mathcal{P})$ and $\mathcal{P}$ be a path system. [REVIEW THIS WHOLE PARA] Then the bijection $\ell : \mathcal{F} \to \mathcal{P}$ induced by this isomorphism is called a *path labeling* of the hypergraph $\mathcal{F}$. To elaborate, let $g : V(\mathcal{F}) \to V(\mathcal{P})$ be the above mentioned isomorphism where $V(\mathcal{F})$ and $V(\mathcal{P})$ are the vertex sets that represent the hyperedges for each hypergraph respectively, $V(\mathcal{F}) = \{v_S \mid S \in \mathcal{F}\}$ and $V(\mathcal{P}) = \{v_P \mid P \in \mathcal{P}\}$. Then the path labeling ℓ is defined as follows: $\ell(S_1) = P_1$ iff $g(v_{S_1}) = v_{P_1}$. Also, the path system $\mathcal{P}$ may be alternatively denoted in terms of $\mathcal{F}$ and ℓ as $\mathcal{F}^\ell$. In most of the scenarios in this paper, what is given is the pair $(\mathcal{F}, \ell)$.
An *overlap graph* $\mathbb{O}(\mathcal{F})$ of a hypergraph $\mathcal{F}$ is a graph such that its vertex set has a bijection to $\mathcal{F}$ and there exists an edge between two of its vertices iff their corresponding hyperedges overlap. Two hyperedges S and S' are said to *overlap*, denoted by $S \between S'$, if they have a non-empty intersection and neither is contained in the other i.e. $S \between S'$ iff $S \cap S' \neq \emptyset, S \nsubseteq S', S' \nsubseteq S$. Thus $\mathbb{O}(\mathcal{F})$ is a subgraph of $\mathbb{I}(\mathcal{F})$ and not necessarily connected. Each connected component of $\mathbb{O}(\mathcal{F})$ is called an *overlap component*.
A path labeling $\ell : \mathcal{F} \to \mathcal{P}$ is defined to be *feasible* if $\mathcal{F} \cong \mathcal{P}$ and this hypergraph isomorphism $\phi : supp(\mathcal{F}) \to supp(\mathcal{P})$ induces a path labeling $\ell_\phi : \mathcal{F} \to \mathcal{P}$ such that $\ell_\phi = \ell$.
A *star* graph is a complete bipartite graph $K_{1,l}$ 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 path from the center to a leaf is called a ray of a k-subdivided star and they are all of length $k + 2$.

3 Characterization of Feasible Tree Path Labeling

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

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

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

The following lemma is useful in characterizing feasible tree path labelings. The proof is in the appendix.

Lemma 1. *If* ℓ *is an ICPPL, and* $S_1, S_2, S_3 \in \mathcal{F}$*, then* $|S_1 \cap (S_2 \setminus S_3)| = |\ell(S_1) \cap (\ell(S_2) \setminus \ell(S_3))|$.

In the remaining part of this section we describe an algorithmic characterization for a feasible tree path labeling. We show that a path labeling is feasible if and only if it is an ICPPL and it successfully passes the filtering algorithms 1 and 2. One direction of this claim is clear: that if a path labeling is feasible, then all intersection cardinalities are preserved, i.e. the path labeling is an ICPPL. Algorithm 1 [a:] has no premature exit condition hence any input will go through it [Prove that the filtered sets has ICPPL iff input PL has ICPPL?]. Algorithm 2 has an exit condition at line 7. It can be easily verified that X cannot be empty if ℓ is a feasible path labeling. The reason is that a feasible path labeling has an associated bijection between $supp(\mathcal{F})$ and $V(T)$ [a:] ~~i.e. $supp(\mathcal{F}^\ell)$~~ such that the sets map to paths, "preserving" the path labeling. The rest of the section is devoted to constructively proving that it is sufficient for a path labeling to be an ICPPL and pass the two filtering algorithms. To describe in brief, the constructive approaches refine an ICPPL iteratively, such that at the end of each iteration we have a "filtered" path labeling, and finally we have a path labeling that defines a family of bijections from $supp(\mathcal{F})$ to $V(T)$ [a:] ~~i.e. $supp(\mathcal{F}^\ell)$~~.
First, we present Algorithm 1 or Filter 1, and prove its correctness. This algorithm refines the path labeling by considering pairs of paths that share a leaf until no two paths in the new path labeling share any leaf.

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

1: $\mathcal{F}_0 \leftarrow \mathcal{F}$, $\ell_0(S) \leftarrow \ell(S)$ for all $S \in \mathcal{F}_0$
2: $j = 1$
3: **while** there is $S_1, S_2 \in \mathcal{F}_{j-1}$ such that $\ell_{j-1}(S_1)$ and $\ell_{j-1}(S_2)$ have a common leaf in T **do**
4: $\quad \mathcal{F}_j \leftarrow (\mathcal{F}_{j-1} \setminus \{S_1, S_2\}) \cup \{S_1 \cap S_2, S_1 \setminus S_2, S_2 \setminus S_1\}$ `Remove` S_1, S_2 `and add the ''filtered'' sets`
5: $\quad$ for all $S \in \mathcal{F}_{j-1}$ such that $S \neq S_1$ and $S \neq S_2$, set $\ell_j(S) \leftarrow \ell_{j-1}(S)$ `Do not change path labeling for any set other than` S_1, S_2
6: $\quad \ell_j(S_1 \cap S_2) \leftarrow \ell_{j-1}(S_1) \cap \ell_{j-1}(S_2)$
7: $\quad \ell_j(S_1 \setminus S_2) \leftarrow \ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2)$
8: $\quad \ell_j(S_2 \setminus S_1) \leftarrow \ell_{j-1}(S_2) \setminus \ell_{j-1}(S_1)$
9: $\quad$ **if** $(\mathcal{F}_j, \ell_j)$ does not satisfy condition (iii) of ICPPL **then**
10: $\quad\quad$ **exit**
11: $\quad$ **end if**
12: $\quad j \leftarrow j + 1$
13: **end while**
14: $\mathcal{F}' \leftarrow \mathcal{F}_j$, $\ell' \leftarrow \ell_j$
15: **return** $(\mathcal{F}', \ell')$

Lemma 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.*

Lemma 3. *In Algorithm 1, at the end of jth iteration, $j \geq 0$, of the* **while** *loop of Algorithm 1, the following invariants are maintained.*

$$
\begin{aligned}
&I \text{ ※ } \ell_j(R) \text{ is a path in } T, && \text{for all } R \in \mathcal{F}_j \\
&II \text{ ※ } |R| = |\ell_j(R)|, && \text{for all } R \in \mathcal{F}_j \\
&III \text{ ※ } |R \cap R'| = |\ell_j(R) \cap \ell_j(R')|, && \text{for all } R, R' \in \mathcal{F}_j \\
&IV \text{ ※ } |R \cap R' \cap R''| = |\ell_j(R) \cap \ell_j(R') \cap \ell_j(R'')|, && \text{for all } R, R', R'' \in \mathcal{F}_j
\end{aligned}
$$

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.
In the base case $(\mathcal{F}_0, \ell_0)$ is an ICPPL, since it is the input. 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 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 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 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 iff the exit condition in line 9, i.e. failure of three way intersection preservation, becomes true in any iteration of the **while** loop. Due to Lemma 3 Invariant IV, the exit condition does not occur if the input is a feasible ICPPL. □

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

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

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

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

Lemma 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 6. *In Algorithm 2, for all $j > 0$, at the end of the jth iteration the four invariants given in lemma 3 are valid.*

Proof. By Lemma 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 5, x is exclusive to S and v is exclusive to $\ell_{j-1}(S)$. Due to this fact, it is clear that the intersection cardinality equations do not change, i.e., invariants II, III, IV remain true. On the other hand, if the input ICPPL is not feasible the invariants are vacuously true. □

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

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

Lemma 7. *If $(\mathcal{F}, \ell)$ is an ICPPL from a tree T and Algorithm 3,* `get-hypergraph-isomorphism` *$(\mathcal{F}, \ell, T)$ returns a non-empty function, then there exists a hypergraph isomorphism $\phi : supp(\mathcal{F}) \to V(T)$ 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 tree passed as input to that recursive call. Moreover, by Lemma 4 and Lemma 5 it is consistent with the tree path labeling ℓ passed. The tree pruning is done by only removing leaves in each call to the function and is done till the tree becomes empty. Thus the returned function $\phi : supp(\mathcal{F}) \to V(T)$ 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. □

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

Theorem 1. *A path labeling $(\mathcal{F}, \ell)$ on tree T is feasible iff it is an ICPPL and Algorithm 3 with $(\mathcal{F}, \ell, T)$ as input returns a non-empty function.*

Proof. From Lemma 7, we know that if $(\mathcal{F}, \ell)$ is an ICPPL and Algorithm 3 with $(\mathcal{F}, \ell, T)$ as input returns a non-empty function, $(\mathcal{F}, \ell)$ is feasible. Now consider the case where $(\mathcal{F}, \ell)$ is feasible. i.e. there exists a hypergraph isomorphism ϕ such that $\ell_\phi = \ell$. Lemma 4 and Lemma 5 show us that filter 1 and filter 2 do not exit if input is feasible. Thus Algorithm 3 returns a non-empty function. □

3.1 ICPPL when given tree is a path

Consider a special case of ICPPL with the following properties.

1. Given tree T is a path. Hence, all path labels are interval labels.
2. Only pairwise intersection cardinality preservation is sufficient. i.e. condition (iii) in ICPPL is not enforced.
3. The filter algorithms do not have **exit** statements.

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

4 Testing for feasible path assignments to k-subdivided star

In this section we consider the problem of assigning paths from a k-subdivided star T to a given set system $\mathcal{F}$. We consider $\mathcal{F}$ for which an associated graph called the overlap graph, denoted by $\mathcal{O}(\mathcal{F})$ is connected. The overlap graph is well-know from the work of [KKLV10,NS09,Hsu02]. We use the notation in [KKLV10]. Hyperedges A and B overlap, written $A \between B$ if A and B have a nonempty intersection but neither of them is contained in the other. The overlap graph $\mathcal{O}(\mathcal{F})$ is a graph in which the vertices correspond to the sets in $\mathcal{F}$, and the vertices corresponding to the

hyperedges A and B are adjacent if and only if they overlap. Note that the intersection graph of $\mathcal{F}$ is different from the $\mathcal{O}(\mathcal{F})$. A connected component of $\mathcal{O}(\mathcal{F})$ 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 $A_1 \in \mathcal{O}_1$ and $A_2 \in \mathcal{O}_2$ are disjoint, or, w.l.o.g, all the sets in $\mathcal{O}_1$ are contained in 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 $\mathcal{O}(\mathcal{F})$ is connected. It is easy to see that once the path assignment for the overlap component in the root is achieved, the path assignment for the other overlap components in the rooted containment tree is essentially finding a path assignment when the target tree is a path: each target path is a path that is allocated to sets in the root overlap compnent. Therefore, for the rest of this section, $\mathcal{O}(\mathcal{F})$ is a connected graph.

We start by recalling a k-subdivided star: Each edge of a star is subdivided k times. Therefore, a k-subdivided star has a central vertex which we call the *root*, and each root to leaf path is refered to as a *ray*. First, we observe that by removing the root from r, we get a collection of vertex disjoint paths of equal length. We refer to the rays as $R_1, \ldots, R_r$, and number of vertices in $R_i, 1 \leq i \leq r$ is $k+2$. Let $v_{i1}, \ldots, v_{i(k+2)} = r$ denote the set of vertices in R_i, and v_{i1} is the leaf.
We will denote the given set system, k-subdivided star and the root of the star by $\mathcal{O}(\mathcal{F})$, T and vertex r, respectively. The algorithm is as follows:runs in two phases. For each hyperedge X, the algorithm maintains a 2-tuple of non-negative numbers $(p_1(X), p_2(X))$. The numbers satisfy the property that $p_1(X)+p_2(X) \leq |X|$, and at the end of the algorithm for each X, $p_1(X)+p_2(X) = |X|$. The meaning is that the algorithm tracks the path length assigned assigned to X from at most two rays. If X gets a path in one ray, then $p_1(X) = |X|$, and $p_2(X) = 0$. In the first phase, it iteratively considers each ray. At the beginning of each iteration hyperedges of $\mathcal{O}$ are classifed into disjoint sets:

1. Those that have been assigned a path which does not contain r in one of the previous iterations.
2. Those that have been assigned two paths containing r from two different rays in two previous iterations.
3. Those that have been assigned one path containing r from a ray in one of the previous iterations. We refer to these as Type-1 hyperedges. For such hyperedges X, $p_1(X)$ denote the length of a sub-path of a ray that has been assigned to X, $p_2(X) = 0$, and a parameter $s(X) = |X| - p_1(X)$ refered to as the *residue* of X that must be assigned a path. Also, it is clear that such a path must start at r and must be in a ray different from the one from a path has already been assigned to X.
4. Those that have not been assigned a path in any previous iteration. We refer to these as Type-2 hyperedges.

In the i-th iteration we refer to the set of hyperedges in the last two classes in the above enumeration as the set $\mathcal{O}_i$. Note that $\mathcal{O}_1 = \mathcal{O}$. In the i-th iteration, hyperedges from $\mathcal{O}_i$ are assigned paths from R_i using the following rules:

1. **Step 1:**
   (a) **There are no type-2 edge:** Select a maximal collection of type-1 hyperedges which form an inclusion chain, and for each such X assign the path in R_i of length $s(X)$starting at $v_{i,k+2} = r$. X is then removed from $\mathcal{O}_i$, and $p_2(X) = s(X)$, and $s(X) = 0$.If any intersection cardinality is violated because of this assignment, EXIT reporting the non-existence of a feasible assignment.

(b) **There are no marginal type-2 edges:** If all type-2 edges of length at most $k+1$, EXIT by saying there is no ICPPL. Otherwise, Find an X of type-2 such that $|X| \geq k+2$, assign it the unique path starting at $v_{i(k+1)} = r$ of length $k+2$, and set $p_1(X) = k+2, s(X) = |X| - (k+2)$, mark this as a type-1 hyperedge, and add it to $\mathcal{O}_{i+1}$ after removing it from $\mathcal{O}_i$. Next, iteratively, consider each type-1 edge $X \in \mathcal{O}_i$, and if it intersects a previously assigned Y such that $\ell(Y) \subseteq R_i$, assign a path of length $s(X)$ containing $v_{i(k+2)} = r$, set $p_2(X) = s(X), s(X) = 0$, and remove X from $\mathcal{O}_i$.

(c) **There are marginal type-2 edges:** Let $X \in \mathcal{O}_i$ be a type-2 hyperedge such that for all $Y \between X$, the overlap sets $X \cap Y$ form a single inclusion chain [KKLV10]. X is refered to as a *marginal hyperedge*. If $|X| \leq k+2$, then X is assigned the path of length $|X|$ starting at v_{i1}, and $p_1(X) = |X|, p_2(X) = 0$. X is removed from O_i and not added to O_{i+1}. If $|X| > k+2$, then $p_1(X) = k+2$, and $s(X) = |X| - (k+2)$. In this case, X is classified as a Type-1 edge, removed from O_i, and added to O_{i+1}.

2. **Step 2:** Iteratively, a hyperedge X is selected from $\mathcal{O}_i$ that has a overlap with one of the hyperedges Y such that $\ell(Y) \in R_i$, and a unique path is assigned to X. The path, say $U(X)$, that is assigned can be decided unambiguously since the $X \between Y$, and all intersection cardinalities can be preserved only at one of the ends of $\ell(Y)$. **Can we put a picture for this**. Let $\ell(X)$ denote the unique path assigned to X. If X is a type-2 hyperedge: an if the unique path of length $|X|$ does not contain r, then $p_1(X) = |X|, p_2(X) = 0$, and X is removed from $\mathcal{O}_i$ and added to $\mathcal{O}_{i+1}$. In the case when $\ell(X)$ has to contain r, then $p_1(X)$ is the length of the path, $p_2(X) = 0$, and $s(X) = |X| - p_1(X)$. Further X is classified as a type-1 hyperedge, added to $\mathcal{O}_{i+1}$ and removed from $\mathcal{O}_i$. In the case when X is a type-1 hyperedge, then we check if $U(X)$, which is of length $s(X)$ contains r. If it does, then we assign $\ell(X) \leftarrow \ell(X) \cup U(X)$, remove X from $\mathcal{O}_i$ and do not add it to $\mathcal{O}_{i+1}$. If not, then we **Exit** reporting that an assignment cannot be found. The iteration ends when no hyperedge in $\mathcal{O}_i$ has an overlap with a hyperedge assigned to R_i.

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

Lemma 8. *Let all hyperedges in $\mathcal{O}_i$ be type-1 edges. Then there is a maximal subset $\mathcal{T}_i \subseteq \mathcal{O}_i$ with the following properties:*

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

Lemma 9. *If there are no marginal type-2 edges in $\mathcal{O}_i$, then there exists at least $r - i$ type-2 hyperedges $X \in \mathcal{O}_i$ such that $|X| \geq k+2$.*

Lemma 10. *At the end of Step-1 in the i-th iteration, if one hyperedge X of type-2 is such that $\ell(X) \subseteq R_i$, then all other hyperedges in $\mathcal{O}_i$ are connected to X in the overlap component.*

Lemma 11. *At the end of Step-2, If control has exit at any time, there is no ICPPL. If control has not exit, then R_i is saturated. No hyperedge of $\mathcal{O}_{i+1}$ will get a path from R_i in the future iterations. No type-2 hyperedge of $\mathcal{O}_{i+1}$ will get a path from R_i. Basically R_i is done.*

Lemma 12. *Finally, we need to prove that the assignment is an ICPPL. Secondly, if there is a permutation then maps sets to paths, then it is indeed an ICPPL, and our algorithm will basically find it. It is a unique assignment upto permutation of the leaves.*

5 Acknowlegements

We thank the anonymous referees of the WG 2011 committee and our colleagues who helped us much with the readability of this document.

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.

[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.

[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.

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

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

[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.

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

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

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

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

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

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

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

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

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

A Appendix

Proof (Proof of Lemma 1). 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 conditions (ii) and (iii) of ICPPL, $|S_1 \cap S_2| - |S_1 \cap S_2 \cap S_3| = |P_1 \cap P_2| - |P_1 \cap P_2 \cap P_3| = |(P_1 \cap P_2) \setminus P_3| = |P_1 \cap (P_2 \setminus P_3)|$. Thus lemma is proven. □

Proof (Proof of Lemma 2). We will prove this by mathematical induction on the number of iterations. The base case $(\mathcal{F}_0, \ell_0)$ is feasible since it is the input itself which is given to be feasible. Assume the lemma is true till $j-1$th iteration. i.e. every hypergraph isomorphism $\phi : supp(\mathcal{F}_{j-1}) \rightarrow V(T)$ that defines $(\mathcal{F}, \ell)$'s feasibility, is such that the induced path labeling on $\mathcal{F}_{j-1}$, $\ell_{\phi[\mathcal{F}_{j-1}]}$ is equal to ℓ_{j-1}. We will prove that ϕ is also the bijection that makes $(\mathcal{F}_j, \ell_j)$ feasible. Note that $supp(\mathcal{F}_{j-1}) = supp(\mathcal{F}_j)$ since the new sets in $\mathcal{F}_j$ are created from basic set operations to the sets in $\mathcal{F}_{j-1}$. For the same reason and ϕ being a bijection, it is clear that when applying the ϕ induced path labeling on $\mathcal{F}_j$, $\ell_{\phi[\mathcal{F}_j]}(S_1 \setminus S_2) = \ell_{\phi[\mathcal{F}_{j-1}]}(S_1) \setminus \ell_{\phi[\mathcal{F}_{j-1}]}(S_2)$. Now observe that $\ell_j(S_1 \setminus S_2) = \ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2) = \ell_{\phi[\mathcal{F}_{j-1}]}(S_1) \setminus \ell_{\phi[\mathcal{F}_{j-1}]}(S_2)$. Thus the induced path labeling $\ell_{\phi[\mathcal{F}_j]} = \ell_j$. Therefore lemma is proven. □