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
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

NOVEMBER 2011

TABLE OF CONTENTS

CHAPTER 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.

Figure 1.1: (i) matrix with COP (ii) without COP

This problem has several practical applications in diverse fields including scheduling [HL06], information retrieval [Kou77b] 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].

1.1 Background

Different aspects of COP were studied as part of this thesis. They are described in the following subsections.

1.1.1 Consecutive Ones Testing

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]. Since then several algorithms have been invented; PQ trees [BL76], variations of PQ trees [MM96b, Hsu01, Hsu02, McC04], ICPIA [NS09] are some of them.

[BL76] showed for the first time that COP testing (COT) is indeed polynomial time solvable by describing the construction of a data structure called PQ trees. This paper 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 procedure in the linear time algorithm described in [BL76].. Since this result, several improvements have been made to the procedural complexity of the algorithm itself and also derivative data structures that are easier to construct. PQR trees is one such data structure [mm96a], [mpt97]. [tm03] describes an improved version of PQR trees. . [h01a] describes the simpler algorithm for COT. Hsu also invented PC trees [h01b] 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 and if not it exited stating negative, this algorithm gives an evidence by way of a certificate of matrix being devoid of COP thereby enabling a user to check the algorithm's result. This is significant from an implementation perspective because automated program verification is hard and manual verification is more viable. Hence having a certificate reinforces its credibility. Note that when the matrix has COP the COP order is the certificate. It may also be noted that the internal machinery of this algorithm is related to the weighted betweenness problem addressed in [Osw97].

To elaborate on what was mentioned earlier about [NS09], the problem of COT can be seen as a constraint satisfaction problem involving a system of sets from a universe as follows. Every column of the binary matrix can be converted into a set of integers which are the indices of rows with 1s in that column. It is apparent that if the matrix has COP, then constructing such sets after applying the COP permutation to the matrix will result in all 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 (set of row indices) which converts each set to its assigned interval. The result in [NS09] characterizes interval assignments to the sets which can be obtained from a single permutation of the rows. They show that for each set, the cardinality of the interval assigned to it must be same as the cardinality of the set, and the intersection cardinality of any two sets must be same as the 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 literature also has results on parallel COT algorithms – [as95] [bs03] [cy91a]

1.1.2 Matrix modification to attain COP or "almost" 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. Nevertheless, the COP is not arbirary and there are several interesting problems when a matrix does not have COP but is "close" to having COP or can be altered to have COP.

– sect 4.1 in [Dom08] has many results surveyed. hardness results, approx. results. results are usually for a class of matrices (a,b) where number columns and rows are restriced to a and b .
– problem of flipping at most k entries of M to make it attain COP. this is NP complete [Boo75]

The first step in this direction is to analyze matrices without COP. [Tuc72] showed that a matrix that does not have COP have certain forbidden substructures that prevent it from having COP. Tucker classified these substructures into five classes of submatrices. This result is presented in the context of AT-free bipartite graphs which is equivalent to matrices with COP. A binary matrix is converted into 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 forbidden substructure. [Dom08] discusses an exact algorithm with time complexity parameterized by the number of ones in the matrix.

Min COS-R (Min COS-C), Max COS-R (Max COS-C) are similarly inspired problems. 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 no-COP matrix M, [Boo75] shows that finding a submatrix M' with all columns but a maxmimum cardinality subset of rows such that M' has COP is NP complete. . [hgXX] 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 minumum number of entries in the matrix that can be flipped 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 ones so that the resulting matrix has COP. As mentioned earlier, this problem had been 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 "near" COP in the storage of very large databases. The problem is that of k-run reordering of binary matrices which they prove to be NP hard using a reduction from the hamiltonian path problem. A connection of COP problem to the travelling salesman problem is also introduced.

The optimization version of the k-run problem, i.e. minimization of number of blocks of ones is proven to be NP complete by [kou77a].

1.1.3 Graph Isomorphism

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 [bl62] on COT are based on the result that interval graphs are AT-free chordal graphs.

1.1.4 Logspace complexity of graph canonization using COP

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 [Lau10a] showed that this implies that it captures PTIME. This result along with that of undirected graph reachability being in logspace [Rei08a], [KKLV10] proved their logspace result.

1.2 Motivation, Objective and Scope

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 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. This generalizes the result in [NS09].

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 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.

1.3 Summary of Results

Given a path labeling ℓ 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 the Section 4, we initiate an exploration of finding feasible path labeling of set systems in a special kind of tree which we call the k-subdivided star. This question is an attempt to generalize the problem of testing if a matrix has the consecutive ones property. However, we restrict the hypergraph $\mathcal{F}$ to be 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.

1.4 Organization of document

Section 2.1 documents the characterization of a feasible path labeling and finally, Section 2.2 describes a polynomial time algorithm to find the tree path labeling of a given set system from a given k-subdivided tree.

CHAPTER 2

RESULTS

2.1 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.

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

The following lemma is useful in subsequent arguments.

Lemma 2.1.1 *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 show that $(\mathcal{F}, \ell)$ is feasible if and only if it is an ICPPL and Algorithm 3 returns a non-empty function. Algorithm 3 recursively does two levels of filtering of $(\mathcal{F}, \ell)$ to make it simpler while retaining the set of isomorphisms, if any, between $\mathcal{F}$ and $\mathcal{F}^{\ell}$.

First, we present Algorithm 1 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 2.1.2 *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 2.1.3 *In Algorithm 1, at the end of jth iteration, $j \geq 0$, of the* **while** *loop, the following invariants are maintained.*

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

1: $\mathcal{F}_0 \leftarrow \mathcal{F}$, $\ell_0(S) \leftarrow \ell(S)$ for all $S \in \mathcal{F}_0$
2: $j \leftarrow 1$
3: **while** there is $S_1, S_2 \in \mathcal{F}_{j-1}$ such that $\ell_{j-1}(S_1)$ and $\ell_{j-1}(S_2)$ have a common leaf in T **do**
4: $\quad \mathcal{F}_j \leftarrow (\mathcal{F}_{j-1} \setminus \{S_1, S_2\}) \cup \{S_1 \cap S_2, S_1 \setminus S_2, S_2 \setminus S_1\}$ ▷ `Remove` S_1, S_2 `and add the "filtered" sets`
5: $\quad$ **for** every $S \in \mathcal{F}_{j-1}$ s.t. $S \neq S_1$ and $S \neq S_2$ **do** $\ell_j(S) \leftarrow \ell_{j-1}(S)$ **end for** ▷ `Carry forward the path labeling for all existing sets 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: $\quad \ell_j(S_1 \setminus S_2) \leftarrow \ell_{j-1}(S_1) \setminus \ell_{j-1}(S_2)$ ▷ `Define path labeling for new sets`
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 (Property 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** $(\mathcal{F}', \ell')$

I $\quad \ell_j(R)$ *is a path in* T, $\qquad$ *for all* $R \in \mathcal{F}_j$

II $\quad |R| = |\ell_j(R)|$, $\qquad$ *for all* $R \in \mathcal{F}_j$

III $\quad |R \cap R'| = |\ell_j(R) \cap \ell_j(R')|$, $\qquad$ *for all* $R, R' \in \mathcal{F}_j$

IV $\quad |R \cap R' \cap R''| = |\ell_j(R) \cap \ell_j(R') \cap \ell_j(R'')|$, $\qquad$ *for all* $R, R', R'' \in \mathcal{F}_j$

Lemma 2.1.4 *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.*

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: $\qquad$ **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 2.1.5 *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.*

Lemma 2.1.6 *In Algorithm 2, for all $j > 0$, at the end of the jth iteration of the* **while** *loop the four invariants given in Lemma 2.1.3 hold.*

We have seen two filtering algorithms above, namely, Algorithm 1 `filter_1` and Algorithm 2 `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 3.

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

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

if T is empty **then**
    return $\emptyset$
end if
$L \leftarrow \{v \mid v \text{ is a leaf in } T\}$
$(\mathcal{F}_1, \ell_1) \leftarrow$ `filter_1`$(\mathcal{F}, \ell, T)$
$(\mathcal{F}_2, \ell_2) \leftarrow$ `filter_2`$(\mathcal{F}_1, \ell_1, T)$
$(\mathcal{F}', \ell') \leftarrow (\mathcal{F}_2, \ell_2)$
$\phi' \leftarrow \emptyset$
for every $v \in L$ **do**
    $\phi'(x) \leftarrow v$ where $x \in \ell_2^{-1}(\{v\})$ ▷ `Copy the leaf labels to a one to one function` $\phi' : supp(\mathcal{F}) \rightarrow L$
    Remove $\{x\}$ and $\{v\}$ from $\mathcal{F}'$, 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 2.1.7 *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 $\phi : supp(\mathcal{F}) \rightarrow V(T)$ such that the ϕ-induced tree path labeling is equal to ℓ or $\ell_\phi = \ell$.*

Theorem 2.1.8 *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.*

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 pair-wise 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.

2.2 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.

2.2.1 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 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.
  $\mathcal{L}_1^i = \{X \mid p_1(X) = |X| \text{ and } p_2(X) = 0 \text{ and } s(X) = 0, X \in O\}$
- $\mathcal{L}_2^i$ *Labeled with r.* Those that have been labeled with two subpaths of $\ell(X)$ containing r from two different rays in two previous iterations.
  $\mathcal{L}_2^i = \{X \mid 0 < p_1(X), p_2(X) < |X| = p_1(X) + p_2(X) \text{ and } s(X) = 0, X \in O\}$
- $\mathcal{T}_1^i$ *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.
  $\mathcal{T}_1^i = \{X \mid 0 < p_1(X) < |X| \text{ and } p_2(X) = 0 \text{ and } s(X) = |X| - p_1(X), X \in O\}$
- $\mathcal{T}_2^i$ *Type 2 / not labeled.* Those that have not been labeled with a path in any previous iteration.
  $\mathcal{T}_2^i = \{X \mid p_1(X) = p_2(X) = 0 \text{ and } s(X) = |X|, X \in O\}$

The set O_i refers to the set of hyperedges $\mathcal{T}_1^i \cup \mathcal{T}_2^i$ in the ith iteration. Note that $O_1 = O$. In the ith iteration, hyperedges from 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 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 O_2 but are added to $\mathcal{L}_1^2$ and all other hyperedges are added to O_2.

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

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

   (b) *Any path of length $s(X)$ at the appropriate end contains r:* If $X \in \mathcal{T}_1^i$ then after finding the appropriate end as in step 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.

REFERENCES

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

[adp80] a, d, and p. Tbd. 1980.

[as95] as. Tbd. 1995.

[bl62] booth and lueker. Tbd. 1962.

[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] b and s. Tbd. 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] c, k, and l. Tbd. 1996.

[cy91a] c and y. Tbd. 1991.

[CY91b] Lin Chen and Yaacov Yesha. Parallel recognition of the consecutive ones property with applications. *J. Algorithms*, 12(3):375–392, 1991.

[df99] d and f. Tbd. 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.

[dgnw06] d, g, n, and w. Tbd. 2006.

[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] Fei. Tbd. 1998.

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

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

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

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

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

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

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

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

[h01a] h. Tbd. 2001.

[h01b] h. Tbd. 2001.

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

[hgXX] h and g. Tbd. XXXX.

[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] j, k, c, k, and v. Tbd: the vldb paper. 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.

[jlm97] j, l, and m. Tbd. 1997.

[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.

[kou77a] kou. Tbd. 1977.

[Kou77b] Lawrence T. Kou. Polynomial complete consecutive information retrieval problems. *SIAM Journal on Computing*, 6(1):67–75, March 1977.

[Lau10a] Laubner. Tbd. 2010.

[Lau10b] Bastian Laubner. Capturing polynomial time on interval graphs. In *LICS*, pages 199–208. IEEE Computer Society, 2010.

[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.

[mm96a] m and m. Tbd. 1996.

[MM96b] 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.

[mpt97] m, p, and t. Tbd. 1997.

[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.

[Osw97] Christof Oswald. Tbd. 1997.

[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.

[Rei08a] Reingold. Tbd. 2008.

[Rei08b] Omer Reingold. Undirected connectivity in log-space. *J. ACM*, 55(4), 2008.

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

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

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

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

[th99] t and h. Tbd. 1999.

[tm03] t and m. Tbd. 2003.

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

[tz03] t and z. Tbd. 2003.

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

[Vel85] Veldhorst. Tbd. 85.

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

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