A Simple Test for the Consecutive Ones Property

Wen-Lian Hsu[1]
Institute of Information Science, Academia Sinica
Nankang, Taipei, Taiwan, ROC
hsu@iis.sinica.edu.tw

ABSTRACT

A (0,1)-matrix satisfies the consecutive ones property if there exists a column permutation such that the ones in each row of the resulting matrix are consecutive. Booth and Lueker [1976] designed a linear time testing algorithm for this property based on a data structure called "*PQ*-trees". This procedure is quite complicated and the linear time amortized analysis is also rather involved. We developed an off-line linear time test for the consecutive ones property without using *PQ*-trees and the corresponding template matching, which is considerably simpler. A simplification of the consecutive ones test will immediately simplify algorithms (and computer codes) for interval graph and planar graph recognition. Our approach is based on a decomposition technique that separates the rows into prime subsets, each of which admits essentially a unique column ordering that realizes the consecutive ones property. The success of this approach is based on finding a good "row ordering" to be tested iteratively.

1. Introduction

A (0,1)-matrix satisfies the *consecutive ones property* (*COP*) for the rows if there exists a column permutation such that the ones in each row of the resulting matrix are consecutive. A consecutive ones test (*COT*) can be used to recognize interval graphs [1,2,3,4] and planar graphs [1,5]. Booth and Lueker [1] invented a data structure called a *PQ*-tree to test the consecutive ones property of (0,1)-matrices in linear time. They associate a *PQ*-tree to each (0,1)-matrix such that each column permutation generated by the *PQ*-tree corresponds to an

[1] The research of this author was supported in part by the National Science Council of the

ordering satisfying the *COP* for the rows. This is implemented in an on-line fashion by adding one row at a time and adjusting the *PQ*-tree accordingly. At each iteration, the algorithm checks nine templates to see if there is a match with part of the current *PQ*-tree. The final *PQ*-tree can represent all possible column permutations that realize the *COP*. However, the on-line version of *PQ*-tree implementation is quite complicated and the linear time amortized analysis is also rather intricate. In this paper we present a decomposition algorithm that does not use *PQ*-trees or any template matching, which is considerably simpler. A simplification of the *COT* will immediately simplify the algorithms (and computer codes) for interval graph and planar graph recognition in [1].

Let M be an $m \times n$ (0,1)-matrix whose total number of ones is r. To each row of M we associate a linked list that consists of all the indices (in the ascending order) of columns which has a non-zero in this row. In such a representation the input size is $O(m+n+r)$. We develop a simple $O(m+n+r)$ time off-line *COT* which adopts a decomposition approach and a set partitioning strategy. The key idea is to determine a "good" ordering of the rows to be tested, which will considerably simplify the process. At the end of the algorithm, we can also obtain a *PQ*-tree that represents all possible column permutations that realizes the *COP*. The idea and implementation of the algorithm is not difficult though the correctness proof is a little complicated.

2. Basic Definitions

Let M be an $m \times n$ (0,1)-matrix whose total number of ones is r. Denote by $R(M)$ the set of rows of M and $C(M)$ the set of columns. Then $|R(M)| = m$ and $|C(M)| = n$. We use u to denote a general row and v to denote a general column. Label the columns in M arbitrarily from 1 to n. Since the algorithm will produce some fractional entries, namely, (1/2)s, the nonzero entries will be either 1 or 1/2. For each row u in $R(M)$, let $CL(u)$ be the set of columns that contain a nonzero in row u. Denote the *size* of a row u by $|CL(u)|$. Label the rows as $u_1, \ldots, u_m$ according to an ascending order of their sizes. For any subset R' of $R(M)$, define $CL(R')$ to be the set of columns that contain a nonzero entry in a row of R'. For each column v in M, let $RW(v)$ be the set of rows that contain a nonzero entry in column v. Denote the *size* of a column v by $|RW(v)|$. For any subset C' of $C(M)$, define $RW(C')$ to be the set of rows that contain a nonzero entry in a column of C'.

Republic of China under grant # NSC 84-2213-E-001-005

Several relationships on the rows: (proper) containment, (strictly) overlapping and independent, are defined below. A row x is said to *be contained in* another row y if no entry of x is greater than the corresponding entry of y. Such containment is said to be *proper* if at least one entry of x is less than the corresponding entry of y. Two rows x and y are said to *overlap* if they have a nonzero entry in a common column. They are said to *overlap strictly* if they overlap but neither is contained in the other. They are said to be *independent* if they do not overlap. The strictly overlapping relationship on the rows plays an important role in the algorithm.

Each column v determines a partition of $R(M)$ into $RW(v)$ and $R(M)$-$RW(v)$. Suppose we refine such a partition based on all the columns one by one, then two rows of M are identical iff they are contained in the same set of the final partition. Since identical rows can be eliminated in this way at the beginning, we shall assume the given matrix M does not contain identical rows. For convenience, we shall not distinguish between the two terms, "row" (resp. "column") and its corresponding "row index" (resp. "column index"). Note that identical columns are allowed.

A *PQ*-tree is a tree whose internal nodes are labeled by either *P* or *Q*, where the children of a *P*-node can be ordered arbitrarily but the children order of a *Q*-node can only be reversed. *PQ*-tree is a useful data structure to represent all possible column permutations that realizes the *COP*. Our decomposition approach will also produce the final *PQ*-tree for the given matrix (without going through the intermediate template operations as described in [1]).

3. Matrix Decomposition

Associate with a matrix M the graph $G(M)$ in which each vertex denotes a row in $R(M)$ and each edge connects a strictly overlapping pair of rows in M. For any subset R' of $R(M)$, denote by $G(R')$ the subgraph of $G(M)$ induced on R'. A subset R' of rows is said to be *connected* if its corresponding induced subgraph $G(R')$ is connected. A submatrix K of M consisting of a subset of rows is said to be *prime* if its associated $G(K)$ is connected. Denote the connected components of $G(M)$ by $G(M_1)$, ..., $G(M_p)$, where M_1, ..., M_p are the corresponding submatrices (called *prime submatrices*) of M. Such decomposition is a decomposition on the rows of $R(M)$.

Assume matrix M does not have identical rows. We shall show that the *COT* for M can be reduced to the *COT* for each of its prime submatrices in $O(m+n+r)$ time. Since each $R(M_i)$ is a subset of $R(M)$, $CL(R(M_i))$ is the set of columns that contain a nonzero entry in a row of M_i.

For convenience, we shall denote $CL(R(M_i))$ by $CL(M_i)$. Define an containment relation ">>" on prime submatrices as follows: we say M_i *contains* M_j, denoted by $M_i >> M_j$, if

(a) $CL(M_i) \supseteq CL(M_j)$

(b) $CL(u) \supseteq CL(M_j)$ for every u in M_i such that $CL(u) \cap CL(M_j) \neq \varnothing$.

Two prime submatrices are *independent* if $CL(M_i) \cap CL(M_j) = \varnothing$. It is easy to see that ">>" is a partial order and the definition of ">>" is independent of the column ordering.

Lemma 3.1. If $CL(M_i) \cap CL(M_j) \neq \varnothing$ for some i and j, then either $M_i >> M_j$ or $M_j >> M_i$.

Proof. Let u_i, u_j be two overlapping rows in M_i, M_j, respectively. Since they belong to different components of $G(M)$, u_j cannot strictly overlap u_i. Without loss of generality, assume u_i contains u_j. All rows of M_j that strictly overlap u_j must also overlap u_i and hence, be contained in u_i. Since $G(M_j)$ is connected, one can argue, through a breadth-first search in graph $G(M_j)$, that all rows in M_j must be contained in u_i. Therefore, we have $CL(u) \supseteq CL(M_j)$ for every u in M_i such that $CL(u) \cap CL(M_j) \neq \varnothing$. Hence, $M_i >> M_j$. Symmetrically, if u_j contains u_i, then following the above argument, we would have $M_j >> M_i$.

Theorem 3.2. *Matrix M satisfies the COP if and only if each of its prime submatrices* M_i, $i = 1, \ldots, p$ *satisfies the COP.*

Proof. The "only if" part is trivial. Hence, assume each M_i satisfies the *COP*. Consider two prime matrices M_i and M_j such that $M_i >> M_j$. From (*b*), columns in $CL(M_j)$ must be identical in M_i. Hence, there must exist a column ordering of M_i realizing the *COP* with columns in $CL(M_j)$ ordered consecutively. In such an ordering, columns of M_i within $CL(M_j)$ can be reordered without affecting the *COP* in M_i. Therefore, one can easily obtain a column ordering of M_i that satisfies the *COP* for both M_i and M_j. In this fashion, we can form a column ordering for M satisfying the *COP* by so ordering each submatrix following a topological order of ">>". ∎

Lemma 3.1 provides an easy way to determine a topological order of ">>". From Theorem 3.2, one can test the *COP* of a matrix M by first finding the connected components of $G(M)$ and then constructing a column ordering satisfying the *COP* for each component. The former will be accomplished in the *G'-CONCTRUCTION* algorithm in Section 5. The latter is relatively simple and is the subject of Section 4.

4. The *COT* for Prime Matrices

Consider a prime matrix M. Let $G(M)$ be its associated connected graph. Let T be a spanning tree of $G(M)$. Apply a preorder traversal on T to obtain an ordering $\pi = (u_1', ..., u_m')$ for rows of M. Then π satisfies that for each i, the induced subgraph on $\{u_1', ..., u_i'\}$ is connected. Thus, for each i, u_i' strictly overlaps with some row in $\{u_1', ..., u_{i-1}'\}$.

A column ordering satisfying the *COP* for M can be easily obtained iteratively by the *COLUMN-PARTITION* algorithm described in Figure 1. In this algorithm, the columns in concern are partitioned into sets, say $S_1, S_2, ..., S_d$, with a fixed left-right ordering (however, the column orders within each set are arbitrary) and are refined at each iteration. Such a partition naturally *induces* a collection of column orderings in which columns in each S_i are arranged consecutively in an arbitrary order and the column groups of the S_is are arranged from left to right according to the order $S_1, S_2, ..., S_d$.

To describe the algorithm we need the following definition. A row u (containing no fractional entries) is said to be *compatible* with a column partition $S_1, S_2, ..., S_d$ with the S_j's ordered from left to right if it satisfies that $CL(u) \cap [S_1 \cup S_2 \cup ... \cup S_d] \neq \varnothing$ and

Either (1) $CL(u) \subseteq [S_1 \cup S_2 \cup ... \cup S_d]$. Let S_{j1} (resp. S_{j2}) be the leftmost (resp. rightmost) set having nonempty intersection with $CL(u)$. Then all sets in between (but excluding) S_{j1} and S_{j2} are contained in $CL(u)$.

Or (2) $CL(u) - [S_1 \cup S_2 \cup ... \cup S_d] \neq \varnothing$. Let S_j be any set having nonempty intersection with $CL(u)$. Then either all sets to the right of S_j are contained in $CL(u)$ or all sets to the left of S_j are contained in $CL(u)$.

The *COLUMN-PARTITION* Algorithm

1. Consider the first two strictly overlapping rows u_1' and u_2'. Partition the columns in $CL(u_1') \cup CL(u_2')$ into three sets $CL(u_1')$-$CL(u_2')$, $CL(u_1') \cap CL(u_2')$ and $CL(u_2')$-$CL(u_1')$, ordered consecutively from left to right. In general, after having considered the rows in $\{u_1', ..., u_{i-1}'\}$, we should obtain a partition, say, $S_1, S_2, ..., S_d$ of columns in $CL(u_1') \cup \ldots \cup CL(u_{i-1}')$, ordered consecutively from left to right. We shall represent each S_k as a tree with the root denoted by RS_k and the column elements of S_k as the children of RS_k.
2. Consider u_i' for $i > 2$. Label all columns in $CL(u_i')$. If a labeled column is a child of a root RS_j , then mark RS_j. Label a subset S_j *full* if all children of RS_j are labeled. Label S_j *partial* if some but not all of its children are labeled. Let S_{j1} (resp. S_{j2}) be the leftmost (resp. rightmost) subset having nonempty intersection with $CL(u_i')$. Check if all sets between (but excluding) S_{j1} and S_{j2} are labeled full. If this is not the case, then u_i' is not compatible

with the partition S_1, S_2, ..., S_d and the algorithm is terminated. Otherwise, consider the following two cases:

(1) If each column in $CL(u_i')$ is a child of some RS_j, then $CL(u_i') \subseteq S_1 \cup S_2 \cup ... \cup S_d$ and u_i' is compatible with the partition S_1, S_2, ..., S_d. If S_{j1} is partial, then split S_{j1} into two subsets S_{j11} and S_{j12} with S_{j12} containing all labeled columns and placed to the right of S_{j11}. To maintain efficiency here, only change the root of those labeled columns in S_{j1}, namely, unlabeled children will keep their original parent and thus, are untouched. Symmetrically, if S_{j1} is partial, split S_{j2} into two subsets S_{j21} and S_{j22} with S_{j21} containing all labeled columns and placed to the left of S_{j22}.

(2) If there exists a column in $CL(u_i')$ that is not a child of any tree, then $S^* = CL(u_i') - [S_1 \cup S_2 \cup ... \cup S_d] \neq \varnothing$. S^* is the set of labeled columns that are not children of any tree. For a new tree with columns in S^* as leaves. Since u_i' strictly overlaps with some row in $\{u_1', ..., u_{i-1}'\}$, we must have $[S_1 \cup S_2 \cup ... \cup S_d] - CL(u) \neq \varnothing$. Therefore, it is impossible that $S_{j1} = S_1$, $S_{j2} = S_d$ and both of S_1 and S_d are full. Now, if neither of the following holds: (i) $S_{j1} = S_1 \subseteq CL(u)$; (ii) $S_{j2} = S_d \subseteq CL(u)$., then u_i' is not compatible with the partition and the algorithm is terminated. Otherwise, in case (i), if S_{j2} is partial, split S_{j2} into two subsets S_{j21} and S_{j22} with S_{j21} containing all labeled columns and placed to the left of S_{j22}. Place S^* to the left of S_1. In case (ii), if S_{j1} is partial, split S_{j1} into two subsets S_{j11} and S_{j12} with S_{j12} containing all labeled columns and placed to the right of S_{j11}. Place S^* to the right of S_d.

3. Reiterate Step 2 with the next row u_{i+1}' until all rows are processed.

Figure 1. The *COLUMN-PARTITION* Algorithm

It is easy to see, in Step 2, that if u_i' is compatible with the partition S_1, S_2, ..., S_d, then, at the end of this iteration, a refinement of S_1, S_2, ..., S_d will be produced in which at most two of the original subsets are split. Also, any column ordering induced by the partition satisfies that the 1s in each row u_k', $k = 1, \ldots, i$ are consecutive. Since Step 2 takes time proportional to $|CL(u_i')|$, the *COLUMN-PARTITION* algorithm takes $O(m+n+r)$ time in total. An example is shown in Figure 2.

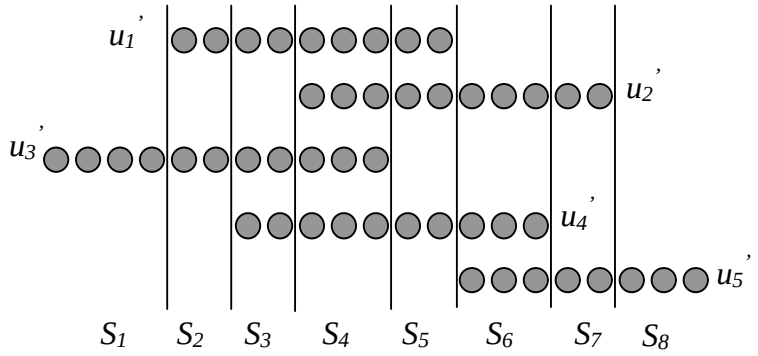

Figure 2. The subsets of the column partition by the rows

Lemma 4.1. *Let M be a prime matrix. Assume the COLUMN-PARTITION algorithm has entered the i-th iteration and u_i' is compatible with the column partition $S_1, S_2, ..., S_d$ of columns in $CL(u_1') \cup \ldots \cup CL(u_{i-1}')$. Then the submatrix M^i of M consisting of rows $u_1', ..., u_i'$ and columns in $CL(u_1') \cup \ldots \cup CL(u_{i-1}')$ satisfies the COP. The partition of columns in $CL(u_1') \cup \ldots \cup CL(u_{i-1}')$ at the end of this iteration is unique up to reversal. Furthermore, any column ordering of M^i satisfying the COP must be induced by such a partition.*

Proof. We prove this by induction. The Lemma is certainly true for u_1' and u_2'. Assume it is true for iterations less than i (>2). By induction, the partition $S_1, S_2, ..., S_d$ of columns in $CL(u_1') \cup \ldots \cup CL(u_{i-1}')$ is unique up to reversal. Consider the i-th iteration. Since u_i' is assumed to be compatible with $S_1, S_2, ..., S_d$, it suffices to consider the two cases in step 2 to compute the new partition. In case (1), S_{j1} and S_{j2} are the only two subsets that would need to be split. In case (2), at most one of S_{j1} and S_{j2} would need to be split. In both cases, it is obvious to see that the splitting operation will yield a unique refinement of $S_1, S_2, ..., S_d$.

By induction, the submatrix M^{i-1} satisfies the *COP* and such a column ordering is induced by the partition $S_1, S_2, ..., S_d$. Since the newly refined partition satisfies that columns in $CL(u_1') \cup \ldots \cup CL(u_{i-1}') \cup CL(u_i')$ are consecutive, the submatrix M^i also satisfies the *COP*. Furthermore, any column ordering of M^i satisfying the *COP* must satisfy that columns in $CL(u_1') \cup \ldots \cup CL(u_{i-1}')$ are induced by the partition $S_1, S_2, ..., S_d$ and that columns in $CL(u_i')$ are consecutive. Hence, such an ordering is induced by the new partition. ■

Theorem 4.2. *A prime matrix M satisfies the COP iff, at every iteration i of the COLUMN-PARTITION Algorithm, u_i' is compatible with the current partition $S_1, S_2, ..., S_d$ of columns in $CL(u_1') \cup \ldots \cup CL(u_{i-1}')$.*

Proof. If u_j' is compatible with the current column partition at every iteration j, then the *COLUMN-PARTITION* algorithm will produce a final partition in which the 1s in each row are consecutive. Hence, M satisfies the *COP*.

Conversely, assume M satisfies the *COP*. We shall prove the theorem by induction on i. Assume the theorem is true for iterations less than i. The *COP* of M implies that the submatrix M^i of M containing rows u_1' , ..., u_i' and columns in $CL(u_1') \cup \ldots \cup CL(u_i')$ must also satisfy the *COP*. By Lemma 4.1, any column ordering of M^{i-1} satisfying the *COP* must be induced by the unique column partition $S_1, S_2, ..., S_d$ of columns in $CL(u_1') \cup \ldots \cup CL(u_{i-1}')$. This means that, to arrange the columns of $CL(u_i')$ consecutively, we cannot change the relative order of the sets $S_1, S_2, ..., S_d$, but could only permute the columns within each subset S_k. Let S_{j1} (resp. S_{j2}) be the leftmost (resp. rightmost) subset having nonempty intersection with $CL(u_i')$. The *COP* of M implies that all sets between (but excluding) S_{j1} and S_{j2} must be labeled full. Now, in case (1), u_i' is compatible with the partition. In case (2), the new subset $S^* = CL(u_i') - [S_1 \cup S_2 \cup ... \cup S_d]$ must be placed either to the left of S_1 or to the right of S_d. Hence the consecutive requirement of columns in $CL(u_i')$ would force one of the following: (i) $S_{j1} = S_1$ and is full; (ii) $S_{j2} = S_d$ and is full. In both cases, we would conclude that u_i' must be compatible with the partition. ■

Note that, in the final unique column partition, all columns contained in the same set must be identical (with respect to rows in M) since these column indices must appear in exactly the same set of rows of M. Suppose the sets of this partition are ordered from left to right as S_1, S_2, ..., S_p. We can construct a *PQ*-tree for the prime matrix M as follows. Construct a *Q*-node (a rectangle) whose children are labeled s_1, s_2, … s_p from left to right representing the sets S_i, $i = 1, \ldots, p$. For each set S_i with $|S_i| > 1$, let the child node s_i be a *P*-node (a circle) with children corresponding to the columns in S_i (all columns in this set must be identical). For each singleton set $S_i = \{w\}$, the node s_i simply represents the column w. An example *PQ*-tree of the prime matrix in Figure 2 is shown in Figure 3.

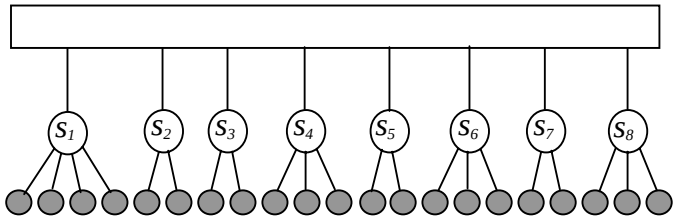

Figure 3. The *PQ*-tree for the matrix in Figure 2

If the given matrix M is not prime, then each of its prime submatrix admits a *PQ*-tree representation as the above. We can combine the *PQ*-trees of the prime submatrices together to form one for M as follows. Perform a topological sort of the prime submatrices under ">>". For each submatrix M_i, define its maximal proper prime submatrices (*MPPS*) as those proper prime submatrices of M_i that are maximal with respect to ">>". Since the columns of each *MPPS* are contained in every row of M_i that overlaps this *MPPS*, these columns must be the children of the same s_i node. For each *MPPS* of M_i, contract its columns into one node representing this *MPPS*. Since each submatrix can only be the *MPPS* of exactly one other submatrix, there will be exactly one node representing it. Now, in a bottom up fashion, recursively substitute the *PQ*-tree of each submatrix into the unique node representing this submatrix. This results in a *PQ*-tree for M.

5. Computing Prime Submatrices

From Section 3, one can identify all prime submatrices of M by computing the connected components of $G(M)$. However, it is too costly to compute all strictly overlapping pairs of rows to find edges of the graph $G(M)$. Instead, for a matrix M satisfying the *COP*, we shall construct a subgraph G' of $G(M)$ with exactly the same connected components in $O(m+n+r)$ time.

We processes the rows from 1 to m. The main iteration of the algorithm is described in the *G'-CONCTRUCTION* algorithm in Figure 4. Since smaller rows are processed before larger ones, we can guarantee that, whenever a row u is being processed, all rows properly contained in u originally have already been processed. During each iteration, we compute a subset of edges between u and its strictly overlapping rows and add them to G'. We shall show that the set of edges computed will eventually yield a spanning subgraph for each $G(M_i)$, where M_i is a prime submatrix of M. If $|CL(u)| > 1$, then at the end of this iteration, all columns of $CL(u)$ are replaced by a special column $[u]$ and the matrix is reduced. This process continues until all rows are processed.

Recall that the input matrix is a linked list $CL(u)$ for each row u. From the list $CL(u)$ for each row u one can construct the linked list $RW(v)$ for each column in $O(m+n+r)$ time. All of the operations in the *G'-CONCTRUCTION* algorithm can be carried out by scanning the linked lists $CL(u)$ and $RW(v)$s where $v \in CL(u)$. For each column v, define $S(v) = \{ w \mid w$ is a row *s.t.* $CL(w) = \{v\} \}$. Set two linked lists, the *potential neighborhood* $PT(u)$ and the *forbidden neighborhood* $FB(u)$ for u, to be $\varnothing$ initially for all u. $FB(u)$ contains those rows that

will not be made adjacent to u and $PT(u)$ contains those rows that will potentially be made adjacent to u (they will be made adjacent to u if they are not in $FB(u)$). $PT(u)$ and $FB(u)$ are represented as linked lists to allow the same row to be appended more than once. However, for convenience, they will sometimes be regarded as sets. Create a graph G' whose vertices are the same as $G(M)$ with no edges initially. In Figure 4, we give the details of the iteration for processing a row u.

The *G'-CONCTRUCTION* Algorithm: Processing a row u

1. Consider the following cases:

   Case 1: $|CL(u)| = 0$, delete row u. Add to graph G' the edges between u and $PT(u)$ - $FB(u)$. Proceed to the next row.

   Case 2: $|CL(u)| = 1$. Let $CL(u) = \{v\}$. Consider the following subcases:

   1.1 v contains no fractional entries. For every row w in $S(v)$, change its v-entry to be 0 (thus, w becomes a row of zeros).

   1.2 v is a special column $[u']$ with fractional entries and the u-entry in column v is 1. Whenever $(u')^A$, $(u')^B$ still have 1/2 in this column (they could have been deleted), add them to $PT(u)$. For every row w in $S(v)$, change its v-entry to be 0.

   1.3 v is a special column $[u']$ and the u-value in column v is 1/2. For any other row w in $S(v)$ with value 1/2 in column v, change its v-entry to be 0.

   Add to graph G' the edges between u and $PT(u)$ - $FB(u)$. Delete u. Proceed to the next row.

   Case 3: $|CL(u)| > 1$. Go to step 2.

2. Mark all columns in $CL(u)$. Determine the linked list $RW(v)$ for each v in $CL(u)$. For each row w in some $RW(v)$ for v in $CL(u)$ (namely, w is a row that overlaps u), count $|CL(w) \cap CL(u)|$.

3. Based on $|CL(w) \cap CL(u)|$ and the values of the corresponding entries in $CL(w) \cap CL(u)$, construct the following sets:

   $D(u) = \{ w \mid CL(w) \neq \varnothing$ and w is properly contained in $u \}$;

   $E(u) = \{ w \mid w$ properly contains $u \}$;

   $I(u) = \{ w \mid w \neq u$ but the entries of w are identical to those of $u \}$

4. Let u^* be a row in $RW(CL(u))$-$D(u)$-$E(u)$-$I(u)$ with the largest $|CL(u^*) \cap CL(u)|$.

5. Let v^* be a column in $CL(u)$-$CL(u^*)$ with the largest $|RW(v^*)$-$E(u)$-$D(u)$-$I(u)|$.

   Let $A(u) = RW(v^*)$-$\{u\}$-$D(u)$-$E(u)$-$I(u)$.

   Let $B(u) = RW(CL(u))$-$A(u)$-$\{u\}$-$D(u)$-$E(u)$-$I(u)$.

Let v^A be a column in $CL(A(u)) \cap CL(u)$ with the largest $|RW(v^A) \cap A(u)|$.

Let v^B be a column in $CL(B(u)) \cap CL(u)$ with the largest $|RW(v^B) \cap B(u)|$.

6. Append $A(u) \cup B(u) \cup D(u)$ to the potential neighborhood $PT(u)$ for u.
   Add to graph G' the edges between u and every row in $PT(u)$ - $FB(u)$.
   Denote the row in $D(u)$ with the largest index by u^D.
   Append u^D to $PT(w)$ for every w in $E(u)$ whose index is smaller than $\pi^{-1}(u^D)$.
   Denote the row in $A(u)$ (resp. $B(u)$) with the smallest index by u^A (resp. u^B).
   Append $[A(u)-\{u\}] \cup \{u^B\}$ to the forbidden neighborhood $FB(u^A)$ for u^A.
   Append $[B(u)-\{u\}] \cup \{u^A\}$ to the forbidden neighborhood $FB(u^B)$ for u^B.
7. For every row w in $D(u) \cup I(u)$, change all of its entries to be 0.
8. At the end of the iteration, delete all columns in $CL(u)$ and create a special column $[u]$, whose entries are 1/2 for rows $\{u^A\}$ and $\{u^B\}$, 1 for rows in $E(u)$, and 0 otherwise. Delete u. After the above column replacement, if a row u' contains only one nonzero entry in a column v', then add u' to $S(v')$. Proceed to the next row.

Figure 4. The *G'-CONCTRUCTION* Algorithm

A row u_i is said to be *made adjacent* (resp. *connected*) to another row u_j in G' if there exists an edge (resp. a path) in G' between u_i and u_j. Steps 3, 4 and 5 merely define some notations. The only steps in which edges of G' are created are Steps 1 and 6. Moreover, a special column $[u]$ is created only when $|CL(u)| > 1$.

We shall prove in Theorem 5.3 that, for a matrix M satisfying the *COP*, the *G'-CONSTRUCTION* algorithm will correctly identify its prime components. Two lemmas are needed for the proof of Theorem 5.3.

Lemma 5.1. *If $u_j \in FB(u_i) \cup PT(u_i)$ with $i < j$, then u_j is made connected to u_i in the G'-CONSTRUCTION algorithm.*

Proof. Suppose u_j is included in $FB(u_i)$ at iteration k ($< i$). Then they both belong to $A(u_k) \cup B(u_k)$ by Step 6. Hence they are connected through u_k.

Now, suppose u_j is included in $PT(u_i)$. If, at iteration i, u_j is in $FB(u_i)$, then by the last paragraph, u_j and u_i are connected through some u_k. Otherwise, $u_j \in PT(u)$ - $FB(u)$ and an edge will be added between u_j and u_i by Step 6. ∎

Lemma 5.2. *If a row u is contained in u' at some iteration but this overlapping relationship is*

later changed before any of CL(u) or CL(u') becomes empty, then u' ∈ FB(*u*) *and u is connected to u' in G'.*

Proof. Let *k* be the first iteration that the overlapping relationship is changed at the end. Then at the end of the *k*-th iteration, there must exists a column *v* in *CL*(*u*)-*CL*(*u'*). Since all columns of $CL(u_k)$ are deleted and only the special column $[u_k]$ is created in Step 8, we must have $v = [u_k]$. Therefore, *u* is u_k^A (or u_k^B) and *u'* belongs to $A(u_k^A)$-$\{u_k^A\}$ (or $B(u_k^A)$-$\{u_k^A\}$). Hence, *u'* ∈ *FB*(*u*). By Lemma 5.1, *u* is connected to *u'* through u_k. ▉

Theorem 5.3. *Suppose M satisfies the COP. Then the graph G' constructed is a subgraph of G(M) and the set of connected components in G' is identical to that of G(M).*

Proof. At each iteration, let $H(u) = \{ w \mid w \text{ is independent of } u \}$. Then the following overlapping relationships remain unchanged by the creation of the special row in Step 9.

1. between $\{u^A,u^B\}$ and rows in $E(u) \cup H(u)$. This is the reason why 1/2 is used.
2. between rows in *E(u)* and *H*(*u*).

Although the overlapping relationships between other sets might be changed, we shall show that they do not change the connected components of *G*(*M*). It suffices to prove the following claims.

Claim 1. *G' is a subgraph of G(M), namely, if* $(u_i,u_j) \notin E(G(M))$ $(i < j)$, *then* $(u_i,u_j) \notin E(G')$.

Proof of Claim 1. Since $(u_i,u_j) \notin E(G(M))$, either they are independent or u_i is contained in u_j originally. If these relationships remain valid when u_i is processed, then the *G'-CONSTRUCTION* algorithm will not make them adjacent (in either Step 1 or 6). Hence, assume these relationships are changed before the *i*-th iteration. Let *k* (< *i*) be the first iteration that these relationships are changed at the end of the iteration. Consider two cases:

1. If they are independent originally, then this change can only occur in Step 8 when $\{u_i,u_j\} = \{u_k^A,u_k^B\}$ for some *k*, in which case u_j will belong to $FB(u_i)$.
2. If u_i is contained in u_j originally, then by Lemma 5.2, $u_j \in FB(u_i)$.

In both cases, they will not be made adjacent inG'. ▉

Claim 2. *If* $(u_i,u_j) \in E(G(M))$, *then they belong to the same connected component in G'.*

Proof of Claim 2. Suppose the claim is false. Let us call any pair of strictly overlapping rows in *M* violating this claim a *bad pair*. Let $\{u_i,u_j\}$ be a *minimal bad pair* satisfying that $i < j$, and for any other bad pair $\{u_p,u_q\}$, we have either (i) $i < p$ or (ii) $i = p$ and $j < q$. We shall prove that the existence of a bad pair will lead to a contradiction.

If u_i overlaps u_j strictly at the *i*-th iteration, then either u_j belongs to $FB(u_i)$ or u_j belongs to $A(u_i) \cup B(u_i)$. In the former case, they are connected in *G'* by Lemma 5.1. In the latter case, u_j

is made adjacent to u_i in Step 6. In both cases they are connected in *G'*, a contradiction.

Hence, the overlapping relationship between u_i and u_j must be changed before the *i*-th iteration. Let *k* (< *i*) be the first iteration that this overlapping relationship is changed at the end of the iteration. By Step 8, we only need to consider the following cases (1) and (2). In each case, we will establish that u_i and u_j are connected in *G'*, contradictory to the assumption that they are a bad pair. Therefore, we could conclude that there exists no bad pair and the theorem is valid.

Case (1). One of u_i and u_j is contained in the other at the end of iteration *k*. Consider two subcases:

Subcase (1.1). u_i and u_j are in $A(u_k)$ (or symmetrically, $B(u_k)$) and each u_j-entry in $CL(u_j)$-$CL(u_k)$ is no greater than the corresponding u_i–entry as shown in Figure 5. But then, u_i and u_j are connected in *G'* through u_k.

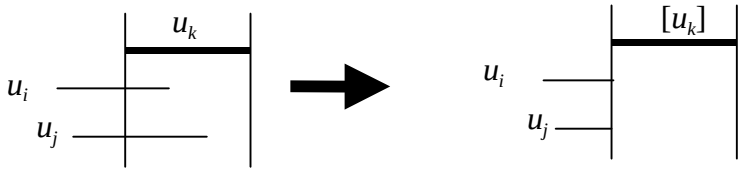

Figure 5. Subcase (1.1) of Claim 2

Subcase (1.2). One of u_i and u_j is a row *z* in $A(u_k)$-$\{u_k^A\}$ (wlog) and the other is an e_A which is a row not overlapping u_k, contained in u_k^A, overlapping with every row in $A(u_k)$ and strictly overlapping *z* at this iteration as shown in Figure 6. Hence, we have $\{z,e_A\} = \{u_i,u_j\}$.

Note that the *COP* of *M* is used to establish the existence of e_A. By the *G'-CONSTRUCTION* algorithm, v^A is contained in $CL(w)$ for every *w* in $A(u_k)$, Since $CL(w)$-$CL(u_k) \neq \varnothing$ for every *w* in $A(u_k)$, by the *COP* of *M*, there must exist a column $(v')^A$ outside of $CL(u_k)$ but belonging to $CL(w)$ for every *w* in $A(u_k)$. Since we assume $CL(z)$-$CL(u_k) \subseteq CL(e_A)$, e_A must contain all such $(v')^A$s. Thus, e_A should overlap with every row in $A(u_k)$.

Since *z* is connected to u_k^A through u_k, we only have to show that u_k^A is connected to e_A in *G'*. Consider the following cases:

i. u_k^A is contained in e_A originally. Since this relationship no longer holds at iteration k, u_k^A must be connected to e_A in *G'* by Lemma 5.2.

ii. e_A is properly contained in u_k^A originally. Then $\pi^{-1}(e_A) < \pi^{-1}(u_k^A) < \pi^{-1}(z)$. At the end of iteration *k*, *z* is contained in e_A. If $CL(e_A) \supseteq CL(z) \neq \varnothing$ when e_A is processed, then *z* will belong to $D(e_A)$ and be adjacent to e_A. Otherwise, at the end of some iteration, say *t* (which is < $\pi^{-1}(e_A)$), $CL(z)$ or $CL(e_A)$ will become empty for the first time.

In any case, we must have $CL(z) = \varnothing$ at the end of iteration *t* and $z \in D(u_t)$. Hence, *z* is

connected to u_t. If $CL(e_A)$ also becomes empty at the end of iteration t, then $e_A \in D(u_t)$ and w, e_A are connected through u_t. If $CL(e_A) \neq \emptyset$, then we have $e_A \in A(u_t) \cup B(u_t) \cup E(u_t)$. If $e_A \in A(u_t) \cup B(u_t) \subseteq PT(u_t)$, by Lemma 5.1, z and e_A are connected through u_t. If $e_A \in E(u_t)$, then the larget indexed row u_t^D of $D(u_t)$ will be added to $PT(e_A)$ by Step 6 (since $\pi^{-1}(e_A) < \pi^{-1}(z) < \pi^{-1}(u_t^D)$). By Lemma 5.1, e_A will be connected to u_t^D. Since u_t^D is connected to u_t, e_A will be connected to z through u_t.

iii. u_k^A is independent of e_A originally. By case 1 in the proof of Claim 1, one of them will belong to the forbidden list of the other. Hence, they are connected in G' by Lemma 5.1.

iv. u_k^A strictly overlaps e_A originally. Suppose u_k^A is not connected to e_A in G'. Then since $\pi^{-1}(u_k^A) < \pi^{-1}(w)$, $\{u_k^A, e_A\}$ would form a bad pair defying the assumption that $\{z, e_A\}$ $(=\{u_i, u_j\})$ is a minimal bad pair, a contradiction.

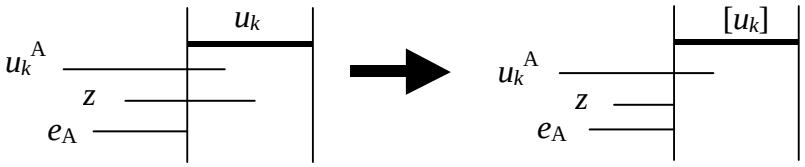

Figure 6. Subcase (1.2) (ii) of Claim 2

Case (2). u_i and u_j are independent at the end of iteration k. There are two subcases:

Subcase (2.1). $u_i \in A(u_k)$, $u_j \in B(u_k)$ and at most one of them can be either u_k^A or u_k^B as shown in Figure 6. Then u_i and u_j are connected in G' through u_k by Lemma 5.1.

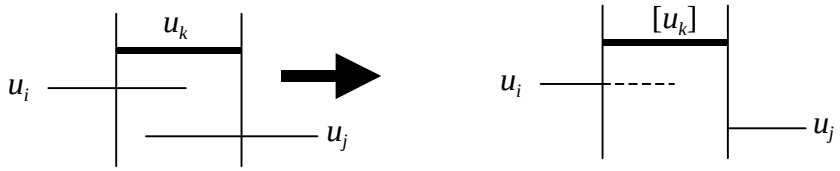

Figure 7. Subcase (2.1) of Claim 2

Subcase (2.2). One of u_i and u_j is a z in $A(u_k)$-$\{u_k^A\}$ (wlog) and the other is a c_B which is a row containing u_k and strictly overlapping u_k^A as shown in Figure 8. Hence, we have $\{z, c_B\} = \{u_i, u_j\}$. Since z is connected to u_k^A through u_k, we only have to show that u_k^A is connected to c_B in G'. Consider two subcases:

i. u_k^A is independent of c_B or one is contained in the other originally. Then, by the discussion of the two cases in the proof of Claim 1, one of them will belong to the forbidden list of the other. Hence, u_k^A is connected to c_B by Lemma 5.1.

ii. u_k^A strictly overlaps c_B originally. Suppose u_k^A is not connected to c_B in G'. Then since $\pi^{-1}(u_k^A) < \pi^{-1}(z)$, $\{u_k^A, c_B\}$ would form a bad pair defying the assumption that $\{z, c_B\}$ $(=\{u_i, u_j\})$ is a minimal bad pair, a contradiction. ∎

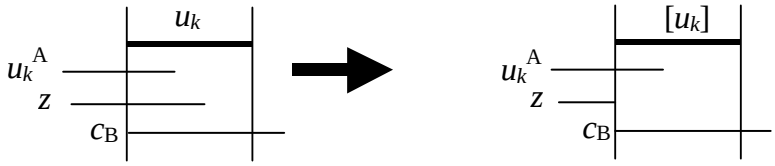

Figure 8. Subcase (2.2) of Claim 2

Although the algorithm will produce a correct spanning subgraph for $G(M)$ when M satisfies the *COP*, we still need to rule out some erroneous subgraphs when the given matrix does not satisfy the *COP*. This can be done by constructing the prime decomposition tree based on the ">>" relation as described in Lemma 3.1.

Define the *maximum row size* of a matrix to be the size of its largest row. Order the connected components in G' as $M_1, \ldots, M_p$ based on their ascending maximum row size. Starting from M_1, let $M_2', \ldots, M_q'$ be those matrices containing a row in $RW(CL(M_1)) - R(M_1)$ arranged in ascending order (recall that $CL(M_i)$ is defined at the bottom of page 3). If M satisfies the *COP*, then we must have $M_q' >> \ldots >> M_2' >> M_1$. Continue this with the remaining matrices until all the ">>" relations have been identified. Now, verify that the ">>" relations are valid in a bottom–up fashion using the containment tree of ">>" through simple column labeling. If there is any violation, matrix M does not satisfy the *COP*.

6. Complexity of the *G'-CONSTRUCTION* Algorithm

In this section we shall show that the complexity of our algorithm is $O(m+n+r)$. There are two main algorithms discussed in this paper: the *COLUMN-PARTITION* algorithm and the *G'-CONSTRUCTION* algorithm. The former takes $O(m+n+r)$ time as discussed in Section 4. Hence, we shall show that the latter also takes $O(m+n+r)$ time.

Lemma 6.1. *If* $|CL(u)| > 1$ *and* $w \in E(u)$*, then every entry of* w *in* $CL(u)$ *must be 1.*

Proof. Suppose some column, say v, of w is 1/2. Since row u is contained in w and its v-entry cannot be 0, its v-entry must be 1/2. But this can only occur when v is a special column created in Step 8 at some iteration k and $\{u,w\} = \{u_k^A, u_k^B\}$. Thus, u and w can share at most one nonzero entry. Since no other step of the *G'-CONSTRUCTION* Algorithm can possibly increase the size of $CL(u) \cap CL(w)$, u and w could share nonzeros in only one column, contradictory to the fact that u and w share nonzeros in every column of $CL(u)$ whose size is at least 2. ∎

Lemma 6.2. *The total number t of new 1s generated during the creation of special columns is no greater than r.*

Proof. Consider the creation of a new representative column $[u]$. By the *G'-CONSTRUCTION* Algorithm, a new column $[u_i]$ is created at iteration i only when $|CL(u_i)| > 1$. Let t_i be the number of new 1s generated in $[u_i]$. Let d_i be the total number of 1s deleted in Step 8 at iteration i. Let d be the total number of 1s ever deleted. Each new 1 is contributed by a row w in $E(u)$. By Lemma 6.1, every entry of w in $CL(u)$ is 1. Hence, for each new 1 generated in $[u_i]$ for w, there are at least two 1s deleted in $CL(w)$, namely, $d_i \geq 2t_i$. Summing up over all iterations, we have $d \geq 2t$. Since $r + t \geq d$, we get $r \geq t$. ∎

Theorem 6.3. *The complexity of the G'-CONSTRUCTION algorithm is* $O(m+n+r)$.

Proof. In Step 1, the set $S(v)$ was constructed in previous iterations. Hence we do not need to scan column v to identify it. Thus, we can skip scanning all zero entries in v.

Next, consider the creation of the special column $[u]$ (recall this can only occur for those u with $|CL(u)| > 1$). Now, all the steps can be accomplished by scanning the linked lists of columns in $CL(u)$. Thus, the zeros in the matrix can be skipped. For example, in Step 3, the calculation of $|CL(w) \cap CL(u)|$ for all w can be done by keeping a counter for each row and add 1 to the counter of a row whenever we scan a column in $CL(u)$ containing a nonzero in this row. Hence, the number of operations is proportional to the total number of nonzeros in the columns of $CL(u)$.

To sum up, the total number of operations in the *G'-CONSTRUCTION* algorithm in all iterations is proportional to the sum of the following numbers:

(a) the number of rows and columns $m + n$.

(b) the total number of nonzero entries in M originally, which is r.

(c) the total number of new 1s created in the special columns, denoted by t.

(d) the total number of (1/2)s generated, which can be at most $2m$.

(e) the total number of elements considered to be included in the potential neighborhood and the forbidden neighborhood of all rows in Step 6 of *The G'-CONSTRUCTION Algorithm*. Note that the same row could be considered more than once for the potential neighborhood or the forbidden neighborhood of another row.

Lemma 6.2 shows that (*c*) is bounded by r. Hence, we are left with the estimation of (*e*). At each iteration i, the total number of elements considered for the potential neighborhood is no greater than $|A(u_i)| + |B(u_i)| + |D(u_i)| + |E(u_i)|$ and that for the forbidden neighborhood is no

greater than $|A(u_i)| + |B(u_i)|$, which together is no more than the total number of nonzeros in the columns of $CL(u_i)$. Therefore, in summary, (e) is no more than the total number of (original and new) ls and (1/2)s, which is (b)+(c)+(d). Hence, the complexity of our algorithm is $O(m+n+r)$. ■

7. Acknowledgment

We are very grateful for many helpful comments of the referee, which have greatly improved the presentation of this paper.

References

1. K.S. Booth and G.S. Lueker, *Testing of the Consecutive Ones Property, Interval graphs, and Graph Planarity Using PQ-Tree Algorithms*, J. Comptr. Syst. Sci. 13, 3 (1976), 335-379.
2. D.R. Fulkerson and O.A. Gross, *Incidence Matrices and Interval Graphs*, Pacific Journal of Math., (1965), 15:835-855.
3. M. C. Golumbic, **Algorithmic Graph Theory and Perfect Graphs,** Academic Press, New York, 1980.
4. N. Korte and R. H. Möhring, *An Incremental Linear-Time Algorithm for Recognizing Interval Graphs*, **SIAM J. Comput**. 18, 1989, 68-81.
5. A. Lempel, S. Even and I. Cederbaum [1967], *An Algorithm for Planarity Testing of Graphs*, **Theory of Graphs**, ed., P. Rosenstiehl, Gordon and Breach, New York, 215-232.