

# PC Trees and Circular-Ones Arrangements

Wen-Lian Hsu \*

Ross M. McConnell †

## Abstract

A 0-1 matrix has the **consecutive-ones** property if its columns can be ordered so that the ones in every row are consecutive. It has the **circular-ones** property if its columns can be ordered so that, in every row, either the ones or the zeros are consecutive. **PQ trees** are used for representing all consecutive-ones orderings of the columns of a matrix that has the consecutive-ones property. We give an analogous structure, called a **PC tree**, for representing all circular-ones orderings of the columns of a matrix that has the circular-ones property. No such representation has been given previously. In contrast to PQ trees, PC trees are unrooted. We obtain a much simpler algorithm for computing PQ trees than those that were previously available, by adding a zero column,  $x$ , to a matrix, computing the PC tree, and then picking the PC tree up by  $x$  to root it.

## 1 Introduction

An **interval graph** is the intersection graph of intervals on a line. That is, there is one vertex for each interval, and two vertices are adjacent if the corresponding intervals intersect.

The problem of recognizing interval graphs has come up in molecular biology. In the late 1950's, before the structure of DNA was well-understood, Seymour Benzer was able to show that the intersection graph of a large number of fragments of genetic material was an interval graph [1]. This was regarded as compelling evidence that genetic information was somehow arranged inside a structure that had a linear topology.

Interval graphs also come up in a variety of other applications, such as scheduling jobs that conflict if they must be carried out during overlapping time intervals. If an interval representation is given, otherwise NP-complete problems, such as finding a maximum clique or minimum coloring, can be solved in linear time [3].

Benzer's work and other applications of interval graphs motivated a search for efficient algorithms for recognizing interval graphs, and for constructing a set of intervals to represent the graph when it is one [8]. A linear-time algorithm was given in 1976 by Booth and Lueker [2].

Booth and Lueker's algorithm works by checking whether a certain 0-1 matrix has the **consecutive-ones property**. A 0-1 matrix has the consecutive-ones property if its columns can be ordered so that, in every row, the ones are consecutive. If such an ordering

---

\*Institute of Information Science, Academia Sinica, Taipei, Hsu@iis.sinica.edu.tw, <http://www.iis.sinica.edu.tw/IASL/hsu/eindex.html>

†Dept. of Computer Science, Colorado State University. rmm@cs.colostate.edu.

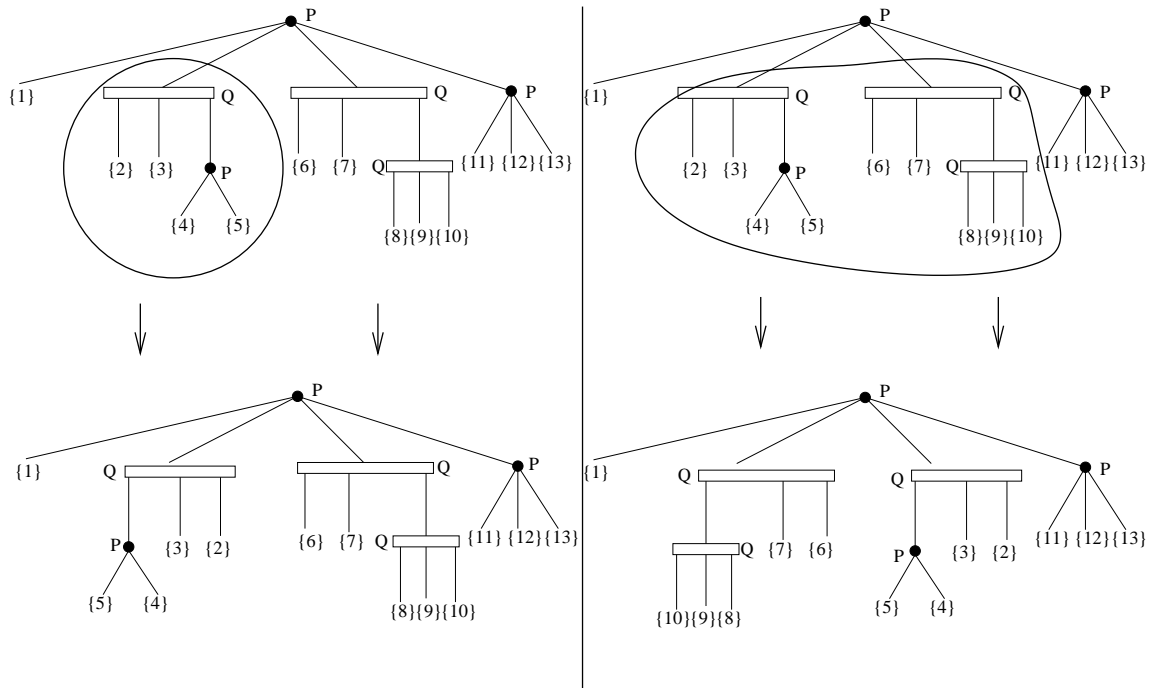


Figure 1: The leaves of a PQ tree are the columns of a consecutive-ones matrix. The left-to-right order of the leaves gives a consecutive-ones arrangement of the columns. So does the result of reversing the leaf descendants of a node. The order of leaves of a consecutive set of children of a P node can also be reversed to obtain a new consecutive-ones arrangement. All consecutive-ones arrangements can be obtained by a sequence of these reversals.

is found, then it is easy to find a representation of the graph with intervals in linear time. The matrix is represented by listing, for each row, the set of columns where a one appears in the row. To obtain a linear-time algorithm for interval-graph recognition, it is necessary to check the matrix in time proportional to the number of ones in it.

To test for the consecutive-ones property, Booth and Lueker developed a representation, called a **PQ tree**, of **all** the consecutive-ones arrangements of the columns. The tree consists of **P nodes** and **Q nodes**. The leaves of the tree are columns of the matrix, and the left-to-right leaf order of the tree gives a consecutive-ones ordering, just as it does when the order of children of a node are reversed, or when the order of children of a P node are permuted arbitrarily (see Figure 1). All consecutive-ones arrangements of the columns can be obtained by a sequence of such rearrangements.

A matrix has the **circular-ones** property if the columns can be ordered so that, in every row, either the zeros are consecutive or the ones are consecutive. That is, it has the circular-ones property if the ones are consecutive when the matrix is wrapped around a vertical cylinder. Booth and Lueker showed that testing for the circular-ones property reduces in linear time to testing for the consecutive-ones property.

Booth and Lueker's algorithm for constructing the PQ tree is notoriously difficult to program. It builds the tree by induction on the number of rows of the matrix. For each row, it must perform a second induction from the leaves toward the root. At each node encountered during this second induction, it uses one of nine **templates** for determining

how the tree must change in the vicinity of the node. Recognizing which template must be used is quite challenging. Each template is actually a representative of a larger class of similar templates, which must be dealt with explicitly by a program.

The literature on problems related to PQ trees is quite extensive. Korte and Moehring [13] considered a modified PQ tree and a simpler incremental update of the tree. Klein and Reif [12] constructed efficient parallel algorithms for manipulating PQ trees. Hsu gave a simple test that is not based on PC trees [11].

In this paper, we give a mathematical characterization of a tree, called a PC tree, for representing the circular-ones arrangements of a matrix. No representation of the set of circular-ones arrangements of a matrix has previously been given. PC trees were originally used as a data structure to find planar embeddings of planar graphs and as a data structure for computing the PQ tree much more simply [19, 10]. It was presented as a rooted tree, but we show here that there are mathematical reasons to treat it as an unrooted tree. One of these is that doing so allows the PC tree to represent all circular-ones arrangements. Though this is an unanticipated use of PC trees, the algorithm for constructing the PC tree that is given in [10] needs no modification in order to achieve this result.

Figure 2 illustrates how the PC tree represents the circular-ones arrangements. The leaves are the columns of the matrix, and are arrayed around the large circle, which represents the circular permutation. The C nodes (double circles) have a cyclic order on their edges that can be reversed. We could think of them as coins with edges attached at discrete points around the sides, and that can be turned heads-up or tails-up. The P nodes (black internal nodes) have no cyclic ordering. The circular-ones arrangements of the columns of the matrix are just those that result from planar embeddings of this gadget that put the leaves on the outer circle. This description makes it obvious what family of circular permutations is represented: you can select an edge and reverse the order of all leaves that lie on one side of the edge, or you can reverse the order of a consecutive set of leaves if they are the leaves of a subset of the trees in the forest that would result from the removal of a P node.

It should be noted that, for the problem of finding planar-graph embeddings, the use of PC trees given by Shih and Hsu in [19] is entirely different from that of PQ trees given by Booth and Lueker modification in [2] to Lempel, Even, and Cederbaum's planarity test [14]. Booth and Lueker used PQ trees to test the consecutive-ones property of all nodes adjacent to the incoming node in their vertex addition algorithm. The leaves of the PQ tree must be those nodes adjacent to the incoming node. Internal nodes of the PQ tree are not the original nodes of the graph. Rather, they are there only to keep track of feasible permutations. In Shih and Hsu's approach, every P node is an original node of the graph, every C node represents a biconnected component in the partial embedding, and nodes adjacent to the incoming node can be scattered anywhere (both as internal nodes and as leaves) in the PC tree. Thus, in Shih and Hsu's approach, a PC tree faithfully represents a partial planar embedding of the given graph and is a more natural representation.

## 2 Preliminaries

If  $X$  and  $Y$  are sets, let  $X\Delta Y$  denote the symmetric difference  $(X - Y) \cup (Y - X)$ .

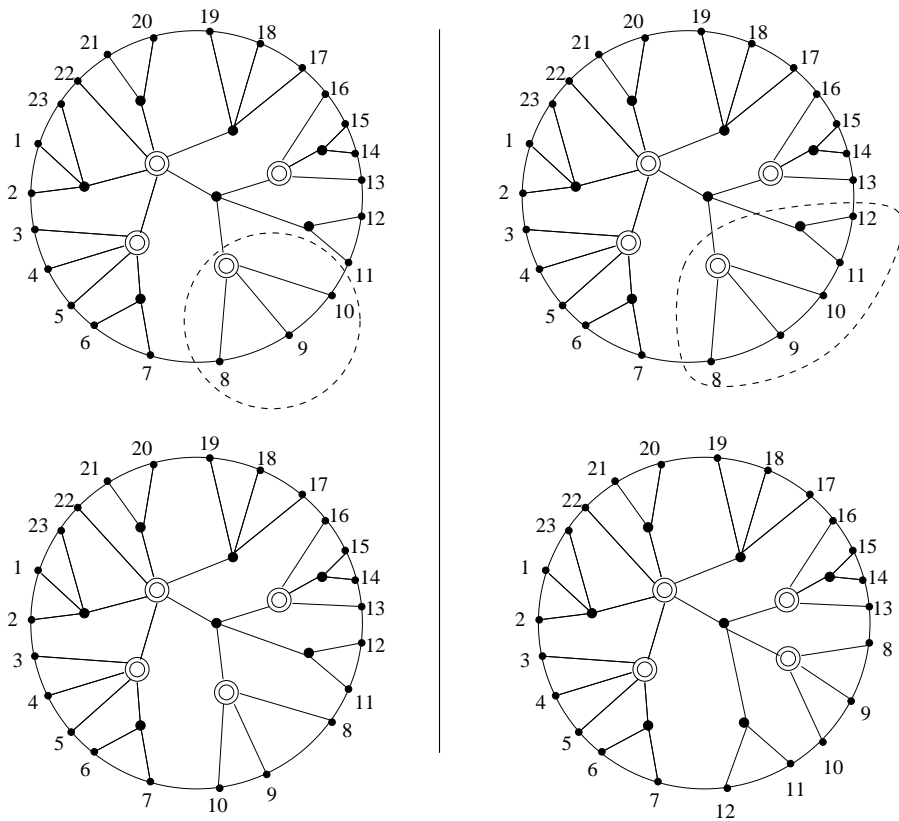


Figure 2: The PC tree can be viewed as a gadget for generating the circular-ones permutations of the columns. The C nodes are represented by double circles and the P nodes are represented by black dots. The subtree lying at one side of an edge can be flipped over to reverse the order of its leaves. The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

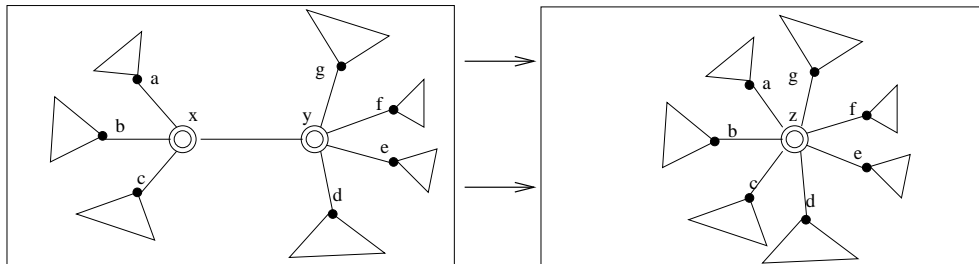


Figure 3: An order-preserving contraction of an edge  $xy$ . The neighbors of  $x$  and  $y$  are cyclically ordered. The edge is removed and  $x$  and  $y$  are identified, so that the cyclic order of neighbors of  $x$  and  $y$  about the edge is preserved.

A **permutation** of a set  $X$  is an ordering of its elements. The **transpose** of a permutation is the ordering obtained by taking the elements in reverse order. Let a **circular permutation** of a set  $X$  be a clockwise ordering of the elements of  $X$  around the circle. The **transpose** of a circular permutation is obtained by exchanging the clockwise ordering and the counter-clockwise ordering.

The **contraction** of an edge  $xy$  in an unrooted tree consists of replacing  $xy$  with a new node  $z$ , whose neighbors are the disjoint union of  $N(x) - \{y\}$  and  $N(y) - \{x\}$ . Since we will deal with unrooted trees whose internal nodes can be cyclically ordered, it will be useful to define the cyclic order of edges incident to  $z$  after the contraction. An **order-preserving contraction** is the one depicted in Figure 3, where the neighbors  $x$  and  $y$  are each consecutive and preserve their original adjacencies in the circular order of  $z$ 's neighbors.

**Definition 2.1** Let a **rooted set family** be a set family  $\mathcal{F}$  on domain  $V$  with the following properties:

1.  $V$  and its singleton subsets are members of  $\mathcal{F}$ ;
2. Whenever  $X, Y \in \mathcal{F}$  and  $X - Y$ ,  $Y - X$ , and  $Y \cap X$  are nonempty, then  $X \cap Y$ ,  $X \cup Y$ , and  $X \Delta Y = (X - Y) \cup (Y - X)$  are also in  $\mathcal{F}$ .

**Theorem 2.2** [17, 6, 7] For a rooted set family  $\mathcal{F}$ , there is a representation with a rooted tree whose nodes are labeled **degenerate** and **prime** and whose leaves are the singleton subsets of  $V$ .  $X \in \mathcal{F}$  iff it is the union of the leaf descendants of a node or the union of the leaf descendants of a set of children of a degenerate node.

Figure 4 gives an example. The best known example of a rooted set family is the set of **modules** of an undirected graph. A set  $X$  is a module if, whenever  $y$  is a vertex that is not in  $X$ ,  $y$  is either a neighbor of every member of  $X$  or of no member of  $X$ . The tree given by Theorem 2.2 is called the **modular decomposition** or **substitution decomposition** of the graph [9, 18]. Other examples of rooted tree families include module-like structures in hypergraphs, matrices, and sets of intervals on a line [17, 6, 7, 15, 16].

**Definition 2.3** Let an **unrooted set family** be a set family  $\mathcal{F}$  on domain  $V$  with the following properties:

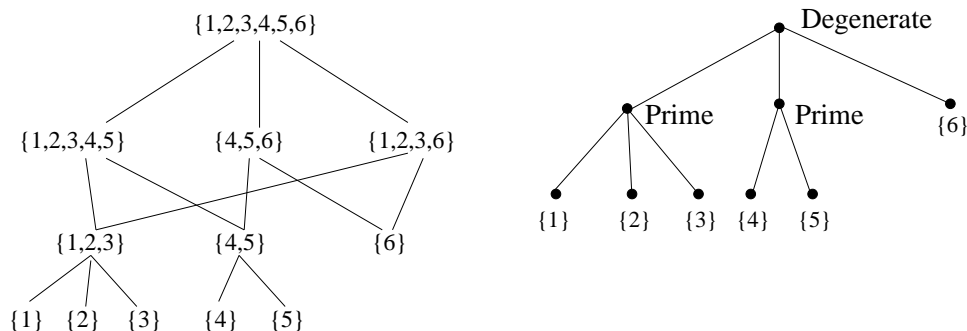


Figure 4: A rooted set family can be represented with a tree whose nodes are labeled **prime** and **degenerate**. A set is in the family if it is the union of leaf descendants of a node, or the union of leaf descendants of a subset of children of a degenerate node.

1. The singleton subsets of  $V$  are members of  $\mathcal{F}$ ;
2. Whenever  $X \in \mathcal{F}$ , then so is  $\overline{X} = V - X$ ;
3. Whenever  $X, Y \in \mathcal{F}$ ,  $X - Y$ ,  $Y - X$ ,  $Y \cap X$  and  $\overline{X \cup Y} = V - (X \cup Y)$  are all nonempty, then  $X \cap Y$ ,  $X \cup Y$ , and  $X \Delta Y$  are also in  $\mathcal{F}$ .

The following is a slight variation of a result given in [4].

**Theorem 2.4** For an unrooted set family, there is an **unrooted** tree whose nodes are labeled **degenerate** and **prime**, and whose leaves are the singleton subsets of  $V$ .  $X \in \mathcal{F}$  iff it is the union of the leaf set of a subtree that results from the deletion of an edge, or the union of leaf sets of some of the subtrees that result from the deletion of a degenerate node.

**Proof:** For fixed  $x \in V$ ,  $\mathcal{F}' = \{Y \mid Y \in \mathcal{F} \text{ and } x \notin Y\}$  is a rooted set family, so it has the rooted-tree representation given by Theorem 2.2. Attaching  $x$  as a child of the root of this tree and turning the tree into an unrooted tree adds the complements of members of  $\mathcal{F}'$  to the set family that is represented. Since the complement of every member of  $\mathcal{F}$  is also in  $\mathcal{F}$ , this adds  $\mathcal{F} - \mathcal{F}'$  to the represented family of sets.  $\square$

Figure 5 gives an example. The best-known example is the **split decomposition** of an undirected graph. The set family that this represents is a family of subsets of the vertices, where  $X$  is in the family iff all vertices not in  $X$  are adjacent to the same subset of  $X$ . Another example are module-like structures in matrices that are invariant under certain types of transformations on the matrices [5]. There the decomposition tree is called a **plane tree**.

### 3 The PC Tree and its relationship to the PQ tree

We view a circular-ones arrangement as a circular permutation of the columns, where the last column is adjacent to the first. Any rotation of this arrangement does not affect the circular-ones property, so there is no reason to specify which column is leftmost.

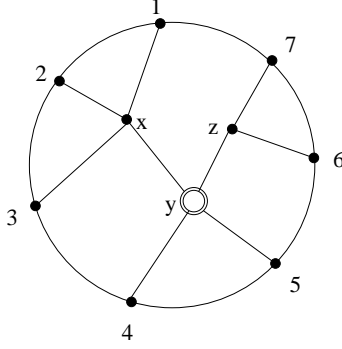


Figure 5: An unrooted set family can be represented with an unrooted tree whose internal nodes are labeled **prime** and **degenerate**. A set is a member of the family if it is the set of leaves of one of the two trees that results when an edge is removed, or the union of the leaves of a subset of the trees that result when a degenerate node is removed. In this case, removal of edges incident to leaves yields the sets  $\{1\}$ ,  $\{2\}$ ,  $\{7\}$  and their complements. Removal of  $xy$  yields  $\{1, 2, 3\}$ , and  $\{4, 5, 6, 7\}$ , and removal of  $yz$  yields  $\{1, 2, 3, 4, 5\}$  and  $\{6, 7\}$ . The fact that  $x$  is a P node also causes  $\{1, 2\}$ ,  $\{2, 3\}$ , and  $\{1, 3\}$ , as well as their complements,  $\{3, 4, 5, 6, 7\}$ ,  $\{1, 4, 5, 6, 7\}$ , and  $\{2, 4, 5, 6, 7\}$  to be in the family of sets represented by the tree. The fact that  $z$  is a degenerate node yields no additional sets, since its degree is only three.

**Definition 3.1** Let  $V$  be the set of columns of a 0-1 matrix.  $X \subseteq V$  is **uniform** in row  $i$  if all elements of row  $i$  in columns of  $X$  have the same value. A **circular module** is any set  $Y$  of columns such that in every row  $i$ , either  $Y$  or  $\overline{Y} = V - Y$  is uniform in row  $i$ . A **linear module** is any set  $Z$  of columns such that in every row  $i$ , either  $Z$  is uniform, or  $\overline{Z}$  is uniformly equal to 0 in row  $i$ .

**Theorem 3.2** The linear modules are a rooted set family.

**Proof:**  $V$  is a linear module because ones are vacuously absent from its complement. A singleton subset of  $V$  is a linear module because any one-element set is uniform.

Suppose  $X$  and  $Y$  are properly overlapping linear modules, *i.e.*,  $X - Y$ ,  $Y - X$  and  $X \cap Y$  are all nonempty. Let  $i$  be an arbitrary row of the matrix.

If both  $X$  and  $Y$  are uniform in row  $i$ , then, since they intersect, so is  $X \cup Y$ , as well as any subset of  $X \cup Y$ , including  $X \cap Y$  and  $X \Delta Y$ . Therefore,  $X \cup Y$ ,  $X \cap Y$  and  $X \Delta Y$  satisfy the requirements of linear modules in row  $i$ .

If exactly one of  $X$  and  $Y$  is uniform in row  $i$ , then suppose without loss of generality that  $X$  is uniform and  $\overline{Y}$  is uniformly equal to zero. Since  $X$  intersects  $\overline{Y}$ , it is uniformly equal to zero.  $\overline{X \cup Y} \subset \overline{Y}$  is uniformly equal to zero.  $X \cap Y \subset X$  is uniformly equal to zero.  $\overline{X \Delta Y} = \overline{X \cup Y} \cup X \cap Y$  is uniformly equal to zero.

If neither  $X$  nor  $Y$  is uniform, then  $\overline{X}$  and  $\overline{Y}$  are uniformly equal to zero. Then  $\overline{X \cap Y} = \overline{X} \cup \overline{Y}$  is uniformly equal to zero, as are all of its subsets, including  $X \Delta Y$  and  $\overline{X \cup Y}$ .

In each case,  $X \cup Y$ ,  $X \cap Y$ , and  $X \Delta Y$  satisfies the requirements of linear modules in row  $i$ . Since  $i$  is arbitrary, they satisfy the requirements in all rows.  $\square$

**Definition 3.3** Let  $\pi$  and  $\tau$  be two orderings of the columns of a matrix. Then  $\pi$  and  $\tau$  are linearly equivalent if one can be obtained from the other by iteratively applying the following step:

- Reverse the order of elements in a linear module that is currently consecutive.

**Theorem 3.4** If  $\pi$  is a consecutive-ones ordering of the columns of a matrix, then the set of consecutive-ones orderings are those that are linearly equivalent to  $\pi$ .

**Proof:** Being a linear module is clearly a necessary and sufficient condition for a consecutive set of columns in a consecutive-ones ordering to be reversed, in order to obtain another consecutive-ones ordering. Therefore, all permutations that are equivalent to  $\pi$  are consecutive-ones orderings. By the PQ tree, every consecutive-ones ordering can be obtained from another by a sequence of such reversals.  $\square$

**Corollary 3.5** The PQ tree is the tree decomposition of the linear modules given by Theorems 2.2 and 3.2, where the prime nodes are the Q nodes and the degenerate nodes are the P nodes.

**Proof:** All transpositions of consecutive sets of columns in a consecutive-ones ordering that the PQ tree permits lie in the set family  $\mathcal{F}$  obtained by interpreting the PQ tree in this way. All linear modules are members of  $\mathcal{F}$ . The set represented by a node is consecutive in every consecutive-ones ordering. The PQ tree can be ordered to make any subset of children of a P node consecutive in a consecutive-ones ordering, so all members of  $\mathcal{F}$  are linear modules.  $\square$

**Theorem 3.6** The circular modules are an unrooted set family.

**Proof:** The singleton subsets of  $V$  are circular modules because they are uniform.

If  $X$  is a circular module, then either  $X$  or its complement is uniform. By symmetry, the complement has the same property.

Suppose  $X$  and  $Y$  are circular modules and  $X \cup Y$ ,  $X - Y$ ,  $X \Delta Y$  and  $V - (X \cup Y)$  are all nonempty. Let  $A = X - Y$ ,  $B = X \cap Y$ ,  $C = Y - X$  and  $D = V - (X \cup Y)$ . Since  $X$  or its complement is uniform, and  $Y$  or its complement is uniform, the union of three of the four sets  $A, B, C, D$  is uniform. Each of  $X \cap Y$ ,  $X \cup Y$ ,  $X \Delta Y$  and  $V - (X \cup Y)$  is either a subset of this union or its complement is.  $\square$

**Definition 3.7** Two circular permutations of the columns of a matrix are **circularly equivalent** if one can be obtained from the other by iteratively applying the following step:

- Reverse the order of elements in a circular module that is currently consecutive.

**Theorem 3.8** The set of circular-ones orderings of a circular-ones matrix are those that are circularly equivalent to some circular-ones ordering.

**Proof:** If we designate a column as  $x_1$ , we may represent any circular-ones ordering by rotating it to give a permutation that begins with  $x_1$ . Let  $\pi$  be such a representation.

Let us complement the rows that have a 1 in column  $x_1$  to get a new matrix  $M'$ . This does nothing to the circular modules or to the circular-ones orderings. It does, however, change the linear modules and the consecutive-ones orderings. The circular modules that do not contain  $x_1$  are now linear modules, and the consecutive-ones orderings where  $x_1$  is first are just the circular-ones orderings with  $x_1$  listed first. Now  $\pi$  is a consecutive-ones ordering, and those consecutive-ones orderings where  $x_1$  is first are those that can be obtained from  $\pi$  by reversing consecutive linear modules that do not contain  $x_1$ , by Theorem 3.4. This is exactly the set of circular-ones orderings that can be obtained from  $\pi$  by reversing circular modules that do not contain  $x_1$ .

The theorem now follows from the fact that any permutation that can be obtained by reversing consecutive circular modules can be obtained by reversing consecutive circular modules that do not contain  $x_1$ , which is seen as follows. Reversing the order of a consecutive circular module  $X$  containing  $x_1$  can be simulated by reversing the order of  $\overline{X}$ , and then reversing the order of the complement of  $\{x_1\}$ . Each of these sets is a consecutive circular module that does not contain  $x_1$ .  $\square$

**Definition 3.9** *Let a PC tree of a consecutive-ones matrix  $M$  be the tree implied by Theorems 2.4 and 3.6, together with the cyclic ordering of the edges incident to prime nodes. The leaves of the tree each correspond to a column of the matrix. The prime nodes are the **C nodes** and the degenerate nodes are the **P nodes**. The cyclic order of edges at each prime node is given by a circular embedding of the graph where the circular permutation of the leaves is some circular-ones ordering of the columns of  $M$ .*

**Theorem 3.10** *A circular permutation of the columns is a consecutive-ones ordering if and only if it can be obtained by transposing the cyclic order of edges at some of the C nodes of the PC tree and embedding the tree in the circle.*

**Proof:** Immediate from Theorems 3.6 and 3.8.  $\square$

**Theorem 3.11** *Computation of the PQ tree reduces in linear time to computation of the PC tree.*

**Proof:** Add a new column  $x$  to the matrix that has all zeros. Compute the PC tree for this new matrix. Then pick up this unrooted tree at  $x$  to make  $x$  the root of a rooted tree. (See Figure 6.) The subtree rooted at the child of  $x$  must be the PQ tree because every circular module that it represents that is not uniform in a column  $i$  has  $x$  in its complement, and therefore satisfies the additional requirement for being a linear module, which is that its complement is uniformly 0 in  $i$ .  $\square$

## 4 Constructing the PC Tree

In this section, we give the algorithm of [10] for constructing the PC tree. The use of circular modules simplifies the analysis, but the algorithm needs no modification in order to match the foregoing definition.

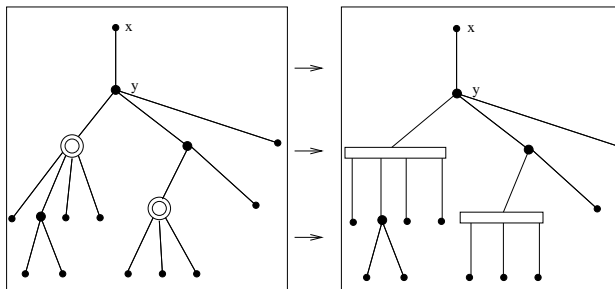


Figure 6: Assigning a new zero column  $x$  to a matrix, computing the PC tree for it, and then picking the PC tree up at  $x$  to root it, gives the PQ tree for the matrix, rooted at  $y$ , when the C nodes are reinterpreted as Q nodes.

A circular module  $X$  is either an **edge module**, which means the leaves on one side of an edge, or a **P module**, which means a set that is not an edge module but the union of leaves in a subset of the trees formed when a P node is removed.

We construct the PC tree by induction on the number of rows of a matrix. The  $i^{\text{th}}$  step of the algorithm modifies the PC tree so that it is correct for the submatrix consisting of the first  $i$  rows of the matrix. As a base case, after the first step, the PC tree consists of two adjacent P nodes, with one of them adjacent to the leaves that correspond to ones in the first row and the other adjacent to the leaves that correspond to the zeros.

During the  $i^{\text{th}}$  step, no new circular modules are created by adding a row, but some circular modules in the first  $i - 1$  rows may become defunct as circular modules once the  $i^{\text{th}}$  row is considered. It is necessary to modify the tree so that it no longer represents these sets as circular modules.

Let the **full leaves** denote the leaves that correspond to ones in row  $i$ , and let the **empty leaves** denote those that correspond to zeros in row  $i$ . If a circular module  $X$  becomes defunct in the  $i^{\text{th}}$  step, then  $X$  and  $\bar{X}$  each contain both empty leaves and full leaves. If  $X$  is an edge module, then the edge that represents it becomes defunct, and must be removed from the tree.

**Lemma 4.1** *If the first  $i$  rows of  $M$  are a circular-ones matrix, then the edges that become defunct in step  $i$  form a path in the PC tree.*

**Proof:** If two edges  $e_a$  and  $e_b$  become defunct, then the two subtrees formed when  $e_a$  is removed each contain both empty and full leaves, and the same is true for  $e_b$ . The removal of any edge on the path from  $e_a$  to  $e_b$  results in two trees, one of which contains one of  $e_a$ 's trees and the other of which contains one of  $e_b$ 's trees. Therefore, every edge on the path from  $e_a$  to  $e_b$  is newly defunct. The set of defunct edges is connected.

Suppose that not all newly defunct edges lie on a path. Then there exist edges  $e_1$ ,  $e_2$  and  $e_3$  such that none of them lies on the path connecting the other two. Of the six sets represented by these three edges, three are disjoint. Each of them contains a mixture of empty leaves and full leaves. By Theorem 3.10, each of these sets is consecutive in any circular-ones permutation, so there are three points on the circle where a transition between zeros and ones occurs in row  $i$  in a circular-ones permutation, a contradiction.  $\square$

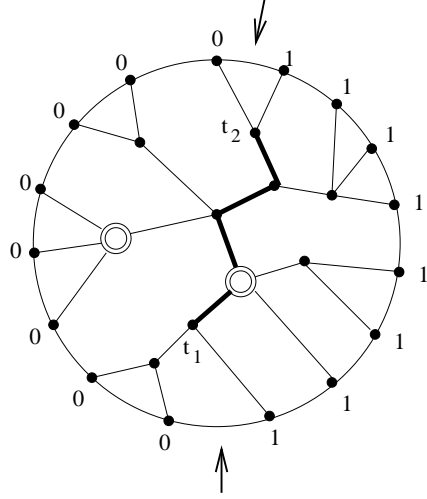


Figure 7: The edges that must be modified when a new row is added are those that represent two sets that have a mixture of zeros and ones, as these sets fail the criterion for being circular modules in the new row. If the matrix has the circular-ones property, these edges lie on a path, called the **terminal path**. The **terminal nodes** are the nodes  $t_1$  and  $t_2$  that lie at the ends of the terminal path.

We may therefore conclude that all alterations to edges must occur on the path of defunct edges in the PC tree (see Figure 7.) Let us call this path the **terminal path**. The **terminal nodes** are the nodes that lie at the ends of the terminal path.

**Lemma 4.2** *The nodes that must be altered in step  $i$  lie on the terminal path.*

**Proof:** Suppose that a node must be altered. If it is a C node, then, since C nodes do not represent any circular modules other than those that are already represented by edges, there is no reason to modify a C node unless it is incident to a defunct edge. Thus, such a C node must lie on the terminal path. A P node  $v$  represents additional P modules that may become defunct. If  $v$  has degree  $d$ , the P modules that it represents are unions of at least two, and at most  $d - 2$  of the leaf sets represented by its incident edges. If one of these,  $X$ , ceases to be a circular module in the  $i^{th}$  step, then both  $X$  and  $\bar{X}$  have a mixture of zeros and ones. At least two of the subtrees formed when  $v$  is removed contain empty leaves, and at least two of them contain full leaves. If one of them has a mixture of these, then the edge from  $v$  to that tree is defunct, and  $v$  lies on the terminal path. A special case occurs when none of the subtrees contain a mixture of zeros and ones, but at least two of them contain only zeros, and at least two contain only ones. In this case, the terminal path has length zero and consists only of  $v$ .  $\square$

We now describe the incremental step of the construction algorithm, which is given in [10].

### Algorithm 4.3 Constructing the PC Tree

*The initial PC tree is a P node that is adjacent to all leaves, which allows all circular permutations.*

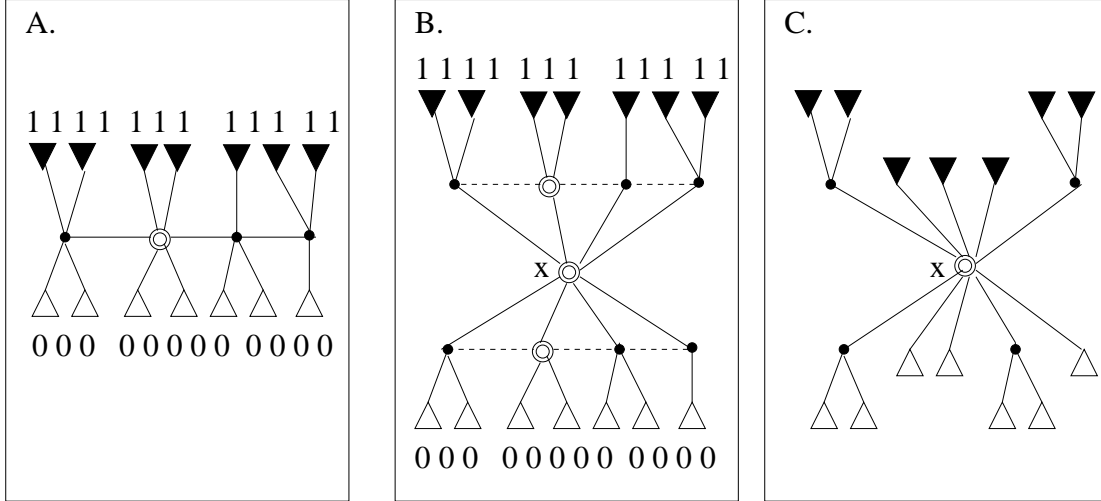


Figure 8: To update the PC tree when a new row is added to the matrix, flip the C nodes and order the P nodes on the terminal path so that the edges that go to trees whose leaves are zeros in the row lie on one side (white) and those that go to trees whose leaves are ones in the row lie on the other side (black). This is always possible if the new matrix has the circular-ones property (Figure A). Then divide each node on the terminal path into two parts, one that is adjacent to the black trees and one that is adjacent to the white trees (Figure B). Replace the edges of these two paths with a new C node,  $x$ , whose cyclic order reflects the order of nodes on these two paths. Finally, contract each edge from  $x$  to a C-node neighbor, and contract each internal node of degree two (Figure C).

At each row:

- Find the terminal path, and then perform flips of C nodes and modify the cyclic order of edges incident to P nodes so that all ones lie on one side of the path (see Figure 8.)
- Split each node on the path into two nodes, one adjacent to the edges to full leaves and one adjacent to the edges to empty leaves.
- Delete the edges of the path and replace them with a new C node  $x$  whose cyclic order preserves the order of the nodes on this path.
- Contract all edges from  $x$  to C-node neighbors, and any node that has only two neighbors.

**Theorem 4.4** Algorithm 4.3 is correct.

**Proof:** Let us assume by induction that  $i \geq 1$  and that the algorithm is correct for the first  $i - 1$  rows. This applies when  $i = 1$  by the base case of the algorithm, which begins with a tree that represents all possible circular permutations.

We must show that after the  $i^{\text{th}}$  update step, the represented circular modules are correct and that the cyclic order of edges at each C node is consistent with some circular-ones arrangement.

Contracting a node of degree two in a planar embedding does nothing to the family of sets represented by the tree or to the cyclic order of the leaves. We may ignore the effect of these contractions.

Let  $T$  be the PC tree before the  $i^{\text{th}}$  row is considered, and let  $T'$  be the PC tree after it is considered.

Those edge sets that cease to be circular modules correspond to edges on the terminal path in  $T$ , by Lemma 4.1, and these edges are deleted, as required. No other edge set of  $T$  ceases to be represented as an edge set in the  $T'$ . We conclude that an edge set of  $T$  is in the set family represented by  $T'$  iff it continues to be a circular module when the new row is considered.

Let  $X$  be a P set that is represented by a P node  $w$  in  $T$ . Suppose  $X$  ceases to be a circular module in the new matrix. By Lemma 4.2,  $w$  lies on the terminal path, and  $X$  and  $\bar{X}$  are non-uniform in row  $i$ . This implies that neither  $X$  nor  $\bar{X}$  lies exclusively on one the full or empty side of the terminal path. Thus,  $X$  ceases to be a represented set when  $w$  is split into two P nodes, one for the full side, and one for the empty side of the terminal path. On the other hand, if  $X$  continues to be a circular module, then either  $X$  or  $\bar{X}$  lies exclusively on either the full or empty side of the terminal path, and it continues to be a union of trees emanating from one of the P nodes that  $w$  is split into. Therefore, either  $X$  or  $\bar{X}$  continues to be represented in  $T'$ , which means that they both are. We conclude that a P set of  $T$  continues to be represented in  $T'$  iff it is a circular module in the first  $i$  rows.

Let us now establish that the cyclic order of C nodes in  $T'$  is correct. After the trees incident to the path are flipped so that the full leaves are on one side and the empty leaves are on the other, they represent a circular-ones arrangement of the first  $i - 1$  rows, since they are permitted by  $T$ . They also represent a circular-ones of the  $i^{\text{th}}$  row, since the ones are consecutive in the cyclic order about the path. Splitting the path does nothing to this cyclic order. Therefore, after the split, the tree represents a circular-ones arrangement of the first  $i$  rows. The cyclic order of all C nodes therefore must be correct.  $\square$

## 4.1 A Data Structure for Representing the PC Tree

For the implementation, we pair up the  $n - 1$  edges with  $n - 1$  of the nodes of the tree, so that each edge is paired with one of the nodes that it is incident to. This can be accomplished by rooting the tree at an arbitrary node in order to define a parent function, and then pairing each nonroot node with its parent edge. It is worth noting that in contrast to the rooting of the PC tree, which serves to give a distinguished role to the root, the sole purpose of this is to make low-level operations more efficient. An example of such an operation is the problem of finding out whether two nodes of an unrooted tree are adjacent, which can be determined in  $O(1)$  time if it is rooted, by examining the parent pointers of the two nodes.

An undirected graph is a special case of a directed graph where every arc  $(u, v)$  has a **twin arc**  $(v, u)$ . Thus, we may speak of the directed *arcs* of the PC tree, not just its edges.

**Definition 4.5** *The data structure for the PC tree is the following. Each P node carries a pointer to the parent edge. Each edge  $uv$  is implemented with two oppositely directed **twin arcs**  $(u, v)$  and  $(v, u)$ . Each arc  $(x, y)$  has a pointer to its two neighbors in the cyclic order about  $y$ , a pointer to its twin, and a bit label that indicates whether  $y$  is the parent of  $x$ . In*

addition, if  $y$  is a  $P$  node, then  $(x, y)$  has a pointer to  $y$ . There is no explicit representation of a  $C$  node; its existence is implicit in the doubly-linked circular list that make up its edges.

If  $(x, y)$  is an arc directed into  $y$  and  $y$  is a  $C$  node, then  $y$  is not represented by an explicit record. We can find  $y$ 's parent edge by cycling through the the records for arcs that are directed into  $y$  in either cyclic direction about  $y$ , until we reach an arc with the required parent bit. Thus, finding a  $C$  node's parent edge is not an  $O(1)$  operation.

The data structure makes no distinction between the two directions in which a list can be traversed; this distinction is made only at the time when a traversal is begun. One must keep track of both the current and previous element. To move to the next element, one must retrieve both neighbors of the current element, and select the one that is different from the previous element.

Using this data structure, it takes  $O(1)$  time to remove or insert a section of a list, given pointers to the endpoints. Since the list draws no distinction between forward and backward, a section of a list can be inserted in either order in  $O(1)$  time.

It therefore supports an order-preserving contraction of an edge between two adjacent  $C$  nodes  $x$  and  $y$  in  $O(1)$  time, given a pointer to  $(x, y)$ , in addition to allowing insertion or removal of an edge from a node's circular adjacency list or reversal of a section of a circular list in  $O(1)$  time

## 4.2 Finding the Terminal Path

Finding the terminal path is the only step where pointers to parent edges are required.

During the  $i^{th}$  step, let us label a leaf as **full** if it corresponds to a column with a one in the  $i^{th}$  row. Let us label an internal node as **full** if all of its neighbors except one have been labeled full. We label an internal node as **partial** if at least one of its neighbors has been labeled full. Whenever we label a node as full, we increment a counter in its non-full neighbor  $x$  that records how many full neighbors  $x$  has, labeling  $x$  as full if this counter rises to one less than the degree of  $x$ . Since every node of the PC tree has degree at least three, the number of full leaves is at least as great as the number of full internal nodes, and there are at most  $k$  full leaves. Assigning full labels takes  $O(k)$  time. The number of partial nodes is at most as great as the number of full nodes, since each full node has at most one partial neighbor. Assigning partial labels takes  $O(k)$  time also.

Let the **apex** of the terminal path be the node on the path that is an ancestor of all others. Note that if  $t$  is a node at the end of the terminal path, then it is a partial node. There may be other partial nodes on the terminal path, but not all nodes on the terminal path are necessarily partial, since they may fail to have full neighbors.

We find the terminal path by starting at each partial node and extending a path up through its ancestors, marking nodes on the path. We do this in parallel at all partial nodes, extending the paths at the same rate. When a path runs into another partial node, we stop extending that path. If a path extends above the apex, we may or may not detect this right away. Eventually, all paths will merge to a connected subtree. The apex is the first point below the highest point of this subtree that is either partial or has two paths entering it. Removing the nodes from the highest point down to the apex, gives the terminal path.

If  $x$  is a node on the terminal path other than the apex, then the parent of  $x$  is also on the terminal path. If  $x$  is a P node, this takes  $O(1)$  time, since it has a pointer to its parent.

If  $x$  is a partial C node then it has a pointer to an edge to a full neighbor. This is always the case at a terminal node, which has no children on the terminal path. If it is on the terminal path but has no full neighbor, then assume that it knows the edge to its child.

The key to bounding the cost of finding  $x$ 's parent when  $x$  is a C node on the terminal path is the observation that if it has any full neighbors, the full neighbors are consecutive, and the edges to its neighbors on the terminal path are adjacent to the full neighbors in the cyclic order. Thus, if it has no full neighbor, we can look at the two neighbors of the child edge in the cyclic order, and one of them must be the parent edge. This takes  $O(1)$  time. If  $x$  has a full neighbor, then we can cycle clockwise and counterclockwise through the edges to full neighbors. Of the first edges clockwise and counter-clockwise that are not to full neighbors, one of these is the parent. In this case, the cost of finding the parent is proportional to the number of full neighbors  $x$ . Thus, we may charge this cost to the full leaves that are on the other side of these nodes from  $x$ .

If  $x$  does not lie on the terminal path, then this procedure may fail, in which case we detect that it is not on the terminal path, or it may succeed, in which case we can bound the cost of finding the parent in the same way.

If the union of all paths traversed has  $p'$  nodes and there are  $k$  ones in row  $i$ , then the total cost is  $O(p' + k)$ . However, the number of nodes in these paths that are not on the terminal path is at most the number of nodes that are on the terminal path, because of the way the paths are extended in parallel. The following summarizes these observations.

**Lemma 4.6** *Finding the terminal path in the  $i^{\text{th}}$  step takes  $O(p + k)$  time, where  $p$  is the length of the terminal path and  $k$  is the number of ones in row  $i$ .*

### 4.3 Performing the update step on the terminal path

The number of full neighbors of nodes on the terminal path is bounded by the number  $k$  of ones in the  $i^{\text{th}}$  row. Before splitting a node, we record its neighbors on the terminal path, and then delete the edges to these neighbors, in  $O(1)$  time. We then split the node by splicing out the full neighbors, and forming a new node with them. The remainder of the old node serves as the other half of the split. This takes time proportional to the number of full neighbors. Since the number of full neighbors of nodes on the terminal path is bounded by the number of ones in the  $i^{\text{th}}$  row, the total time for this is  $O(p + k)$ . Creating the new C node  $x$  and installing edges to the split nodes in the required cyclic order then takes  $O(p)$  time.

Since our data structure includes a parent function, we must assign the parent bits to the new edges. Let  $y$  be the copy of the apex that retains its parent edge after the split of the apex. Except in the case of the apex, the parent edge of every vertex on the terminal path is an edge of the terminal path, and these edges have been deleted. Thus, none of these nodes have a parent edge. We make  $x$  the new parent of these nodes, and we let  $y$  be the parent of  $x$ . This takes  $O(p)$  time by setting the appropriate bits in the  $O(p)$  edges incident to  $x$ .

The operation of deleting a C node  $z$  from  $x$ 's neighborhood and replacing it with the neighbors of  $z$  is just a contraction of the edge  $xz$ , which we have shown takes  $O(1)$  time. These observations can be summarized as follows:

**Lemma 4.7** *Updating the tree during the  $i^{\text{th}}$  step takes  $O(p+k)$  time, where  $p$  is the length of the terminal path and  $k$  is the number of ones in row  $i$ .*

#### 4.4 The linear time bound

The algorithm processes one row at a time of the matrix  $M$ . Let  $T$  denote the current state of the PC tree after the first  $i$  rows have been processed. That is,  $T$  is the PC tree for the submatrix induced by the first  $i$  rows. Let  $C_i$  be the set of C nodes and let  $P_i$  be the set of  $P_i$  nodes in  $T$ , and let  $u_i$  be the number of ones in the rows of the matrix that have not yet been processed. If  $x$  is a node of  $T$ , let  $\text{deg}(x)$  denote the number of neighbors of  $x$  in  $T$ .

We adopt a budget where we keep an account that must have a number of credits  $\phi(M, i)$  in reserve, where  $\phi(N, i)$  is given by the following:

- $\phi(M, i) = u_i + |C_i| + \sum_{x \in P_i} (\text{deg}(x) - 1)$ .

Each credit can pay an  $O(1)$  operation. Any operation that reduces  $\phi$  allows us to withdraw credits from the account to pay for some of the operations. Since  $\phi(M, 0)$  is  $\Theta(m)$ , we must pre-pay the cost of later operations by making a deposit of  $\Theta(m)$  credits to the account. If we can maintain this reserve as we progress and still pay for all operations with withdrawals, then, since  $\phi$  never runs the account to zero, the running time of the algorithm is  $O(m)$ .

To cover the costs in row  $i$ , we must be able to withdraw  $k + p$  credits, by Lemmas 4.6 and 4.7. Of these,  $k$  credits are justified by processing ones in the row, since this reduces the number of unprocessed ones. Each C node on the terminal path is first split, but then both of these copies are contracted. This decreases the number of C nodes by one without changing the degrees of any P nodes, so spending a credit for each C node on the path is within the budget. Suppose a P node does not lie at the end of the terminal path. If it is split, the sum of degrees of the two parts is the same as the degree of the original, after the terminal-path edges are deleted and replaced with an edge to the new C node. However, in calculating  $\phi$ , subtracting one from the degrees of two P nodes instead of one frees a credit. If the P node has only empty neighbors or only full neighbors, it is not split. In this case its degree decreases by one when its two incident terminal-path edges are deleted and replaced by a single edge to the new C node. A P node at the end of the terminal path fails to free up a credit, but there are only  $O(1)$  of these. Contractions of nodes of degree two free up a credit, whether they are C nodes or P nodes.  $\Theta(k + p)$  credits for row  $i$  can be paid out while adhering to the budget.

## 5 Conclusions

We have shown that if we add a zero column to the matrix, the PC tree is the PQ tree without the root. Moreover, we build the PC tree row-by-row in the matrix, so it must be

the case that after each row is processed, the PC tree we have so far is the unrooted PQ tree for the rows we have processed. Booth and Lueker's algorithm also works row-by-row, building the PQ tree for the rows that have been processed so far.

This implies that the update step for the PC tree must actually do exactly what Booth and Lueker's algorithm does, except that it fails to keep track of where the root is. This is demonstrated explicitly for each of the templates in [10].

One interpretation of this is that the large number of templates used by Booth and Lueker algorithm is an artifact of the unnecessary introduction of the notion of a root into the analysis. The many templates are an iterated reanalysis of a simple situation, under all possible assumptions about where this unnecessary root might lie.

Instead of the credit invariant that we use in the analysis of our time bound, Booth and Lueker use one where the account has a credit for each unprocessed one in the matrix, a credit for each C node, and a credit for each node that has a parent that is a P node. It is not hard to see that these are equivalent, but, by formulating it in this way, we have shown that the root is also irrelevant to the analysis of the time bound of either algorithm.

There is no real reason to root this PC tree once it is constructed to obtain a PQ tree. The circular-ones permutations that it represents in this case are just the consecutive-ones permutations of the original matrix, where the linear order of the leaves is read off by starting from the additional zero column and traveling around the circle until this column is encountered again. The PC tree solves a more general problem. Rooting it is just an extra step that ensures that it only solves a special case of the problem.

## References

- [1] S. Benzer. On the topology of the genetic fine structure. *Proc. Nat. Acad. Sci. U.S.A.*, 45:1607–1620, 1959.
- [2] S. Booth and S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *J. Comput. Syst. Sci.*, 13:335–379, 1976.
- [3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw Hill, Boston, 2001.
- [4] W. H. Cunningham. Decomposition of directed graphs. *SIAM J. Algebraic Discrete Methods*, 3:214–228, 1982.
- [5] A. Ehrenfeucht, T. Harju, and G. Rozenberg. *The Theory of Two-Structures*. World Scientific, Singapore, 1999.
- [6] A. Ehrenfeucht and G. Rozenberg. Theory of 2-structures, part 1: Clans, basic subclasses, and morphisms. *Theoretical Computer Science*, 70:277–303, 1990.
- [7] A. Ehrenfeucht and G. Rozenberg. Theory of 2-structures, part 2: Representations through labeled tree families. *Theoretical Computer Science*, 70:305–342, 1990.
- [8] D.R. Fulkerson and O. Gross. Incidence matrices and interval graphs. *Pacific J. Math.*, 15:835–855, 1965.

- [9] T. Gallai. Transitiv orientierbare Graphen. *Acta Math. Acad. Sci. Hungar.*, 18:25–66, 1967.
- [10] W.L. Hsu. PC-trees vs. PQ-trees. *Lecture Notes in Computer Science*, 2108:207–217, 2001.
- [11] W.L. Hsu. A simple test for the consecutive ones property. *Journal of Algorithms*, 42:1–16, 2002.
- [12] P.N. Klein and J.H. Reif. An efficient parallel algorithm for planarity. *Journal of Computer and System Science*, 18:190–246, 1988.
- [13] N. Korte and R.H. Möhring. An incremental linear-time algorithm for recognizing interval graphs. *SIAM J. Comput.*, pages 68–81, 1989.
- [14] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In P. Rosenstiehl, editor, *Theory of Graphs*, pages 215–232. Gordon and Breach, New York, 1967.
- [15] R. M. McConnell. Linear-time recognition of circular-arc graphs. *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS01)*, 42:386–394, 2001.
- [16] R.M. McConnell and J.P. Spinrad. Construction of probe interval models. *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 866–875, 2002.
- [17] R. H. Möhring. Algorithmic aspects of the substitution decomposition in optimization over relations, set systems and boolean functions. *Annals of Operations Research*, 4:195–225, 1985.
- [18] R. H. Möhring and F. J. Radermacher. Substitution decomposition for discrete structures and connections with combinatorial optimization. *Annals of Discrete Mathematics*, 19:257–356, 1984.
- [19] W.K. Shih and W.L. Hsu. A new planarity test. *Theoretical Computer Science*, 223:179–191, 1999.