# Implementation of the PC-Tree Data Structure Project Proposal

Michael Luk and Michael Zhou
*CPSC 445, University of British Columbia*

February 27, 2014

## 1 Introduction

The PQ-tree data structure can represent consecutive-ones ordering of a 0-1 matrix. This has various applications in genomics. For instance, in the reconstruction of ancestral genomes [1], the orderings of a PQ-tree can provide the different permutations of a conserved interval, used for representing potential ancestors of a set of genomes. The *PC-tree* data structure is a similar structure, used for representing a *circular-ones* ordering of a 0-1 matrix. Compared to PQ-trees, PC-trees are implemented with a simpler algorithm, despite being highly similar. As part of a course project, we are implementing the PC-tree data structure as described by Hsu et al. [2]. In this project proposal, we will briefly go over the theory behinds PC-trees, as well as explain its algorithm and implementation details.

## 2 Project Objective

Implement the PC-tree data structure as explained by Hsu et al. [2].

## 3 Theory

This section briefly outlines the theory behind PC-trees. Additional detail can be found in [2].

A 0-1 matrix is a matrix where every element is either a 0 or 1. A 0-1 matrix has the consecutive-ones property if the columns of the matrix can be arranged so that in every row the ones of the row are consecutive. Matrices can be represented as a PQ-tree, which can be used to obtain the consecutive-ones ordering of a matrix. A 0-1 matrix has the *circular-ones* property if the columns can be rearranged so that the either the ones or the zeroes of the rows are consecutive. Existing algorithms for constructing PQ-trees are difficult to implement. An alternative tree, the PC-tree, can be used to represent circular-ones arrangement of a matrix.

In a PC-tree, the leaves of the tree represent the columns of a matrix. C-nodes are nodes whose childs have a cyclic order that can be reversed, while P-nodes have no cyclic ordering. The circular-ones arrangements of a matrix can be obtained from all possible reversals of edges and leaves. The transpose of a permutation is obtained by reversing the order of the set. Trees that are unrooted can be contracted, where the nodes $xy$ are replaced with the new node $z$. The contraction is *order-preserving* if the original adjacencies are preserved in $z$'s circular order.

## 4 Proposed Method

This section will outline the algorithm and implementation of the PC-tree structure.

### 4.1 Algorithm

Given a 0-1 matrix $M$, its PC-tree $T$ is constructed going through $M$ one row at a time. Initially, $T$ consists of a P-node adjacent to all leaves. Each leaf represents a column of $M$. For each row of $M$, we perform the following on the current tree $T$:

1. Find the terminal path. Perform transformations to the nodes of this path such that all 1-leaves (full) are on one side of the terminal path, and all 0-leaves (empty) are on the other.

2. Split the nodes of the terminal path into two, such that one is adjacent to the edges of 1-leaves, and the other to 0-leaves.

3. Delete the edges of the terminal path, and add a new C-node $x$ such that the cyclic order preserves the order of the nodes of the path.

4. Contract all edges from $x$ to C-node neighbours, and nodes with only two neighbours.

After iteration through all rows, we obtain our final $T$, a PC-tree representing all orderings of $M$ with the circular-ones property.

## 4.2 Implementation Details

### 4.2.1 Data Structure

A PC-tree consists of nodes and edges. An edge $uv$ is represented with twin arcs $(u, v)$ and $(v, u)$. Each arc $(x, y)$ contains the following:

1. a pointer to its twin arc

2. pointers to the two neighbours in the cyclic order about $y$

3. a bit label to indicate if $y$ is parent of $x$

4. a pointer to $y$ only if $y$ is a P-node

Additionally, P-nodes contain a pointer to its parent edge.

This data structure for representing edges allows us contract adjacent C-nodes $x$ and $y$ given the pointer to $(x, y)$, insert/remove an edge from a node's circular adjacency list, and reverse a section of a circular list all in constant time. We will implement PC-tree as a class, with methods for performing each step of the algorithm.

### 4.2.2 Terminal Path

Finding to terminal path is not trivial. This section describes the procedure in finding the path.

To obtain the terminal path, we first must label all internal nodes as full, partial or empty. All leaf nodes are considered *full*. An internal node is *full* if all neighbours but one are labeled full. Otherwise, if at least one neighbour is full, we label it *partial*. All other cases will be considered *empty*.

After labelling, we perform the following:

1. starting at each partial node, extend up the paths through its ancestors, while marking them.

2. once a path runs into another partial node, stop extending that path.

3. once all paths stop extending, we get a subtree connecting all paths

4. locate the *apex*, which is the first point below the highest point of this subtree that is partial or has two paths entering it

5. remove all nodes from the highest point of this subtree, to the apex.

The remaining marked nodes will then represent the terminal path.

# 5 Team Responsibilities

Both team members will be involved in the implementation of the PC-tree structure.

# 6 Conclusion

There is a wide range of applications for PC/PQ-Trees in bioinformatics. Previous projects have used PQ-trees to reconstruct gene orders of ancestral genomes. Using PC-trees instead of PQ-trees may prove to be simpler to implement and have better performance. Comparison with current programs using PQ-trees can be done for validation. This project proposal has outlined the algorithm and implementation detail needed to implement the PC-tree data structure.

# References

[1] Bergeron, A., Blanchette, M., Chateau, A., Chauve, C., *Reconstructing Ancestral Gene Orders Using Conserved Intervals*, Proc. of Workshop Algorithms in Bioinformatics (WABI), 14-25, 2004.

[2] Hsu, W., McConnell, R., *PC trees and circular-ones arrangements*, Theoretical Computer Science 296:1, 99-116, 2003