Characterizing and Recognizing Path Graphs and Directed Path Graphs using PR-trees

Steven Chaplick

Department of Computer Science, University of Toronto

Abstract

The ***PR-tree*** data structure is introduced to characterize the sets of path-tree models of path graphs. We further characterize the sets of directed path-tree models of directed path graphs with a slightly modified PR-tree. An algorithm is presented which constructs a PR-tree that captures the path-tree models of a given graph. This algorithm is extended to construct the modified PR-tree which captures the directed path-tree models of a given graph. For a given graph with n vertices and m edges, these algorithms have a runtime complexity of $\mathcal{O}(a(n+m)*n*m)$ (where $a(s)$ is the inverse of Ackermann's function arising from Union-Find[22]). Additionally, the size of a PR-tree produced by these algorithms is $\mathcal{O}(n+m)$. These algorithms are further used to recognize path graphs and directed path graphs. This approach nearly matches the timing of best known recognition algorithms for both path graphs and directed path graphs (both of which are $\mathcal{O}(n*m)$).

Key words: PR-tree, intersection graphs, path graphs, directed path graphs, clique trees

1 Introduction

All graphs considered here are finite, have no parallel edges, and no loops. For a graph G, we use n and m to denote the number of vertices and edges of G respectively. Also, when discussing the intersection of subgraphs of a graph we mean vertex (rather than edge) set intersection.

The following notation is used in this paper. Given a subset V' of V, $\boldsymbol{G[V']}$, is the subgraph of G ***induced*** by V'. A ***clique*** C of G is a subset of V where every pair is adjacent. Let $\mathcal{K}_G$ denote the set of maximal cliques of G, and

Email address: `chaplick@cs.toronto.edu` (Steven Chaplick).
URL: `http://www.cs.toronto.edu/˜ chaplick` (Steven Chaplick).

for every vertex $v \in V$, we let S_v denote the set of maximal cliques of G that contain v (i.e., $S_v = \{C : C \in \mathcal{K}_G \text{ and } v \in C\}$). For a vertex v of G, the ***neighbourhood of v*** $N(v)$ is the set of vertices adjacent to v. We also use $[i, j]$ to denote the set $\{i, i+1, ..., j-1, j\}$ for natural numbers i and j where $i \leq j$.

A ***tree*** is a connected graph with no cycles. Furthermore, a ***directed tree*** is a tree where each edge is treated as an ordered pair. Notice that, in a directed tree, we can have multiple vertices with no edges that end at them (i.e., multiple "source vertices" are possible). A ***rooted tree*** T is a directed tree with a single source vertex (note: since T is a directed tree, there must be at least one source). A ***directed path*** is a rooted tree with a single leaf (i.e., a path whose edges are all oriented in the same direction). Finally, an ***ordered tree*** is a rooted tree where the children of every vertex are ordered.

A graph is ***chordal*** when it has no induced k-cycle, for $k \geq 4$. In [9] Gavril showed that the chordal graphs are the intersection graphs [1] of ***subtrees of a tree***. In particular, a graph G with $V(G) = \{v_0, v_1, ..., v_{n-1}\}$, is chordal iff:

There is a *tree* T, and a collection $\{t_0, t_1, ..., t_{n-1}\}$ of *subtrees* of T, such that: $T = \bigcup_{i=0}^{n-1} t_i$ and $\forall i, j \in [0, n-1], v_i v_j \in E(G) \Leftrightarrow V(t_i) \cap V(t_j) \neq \emptyset$.

We call such a tree and collection of subtrees a ***tree-tree model*** of G.

An ***interval*** graph is the intersection graph of a set of intervals on the real line; equivalently, it is the intersection graph of ***subpaths of a path***. In particular, a graph G with $V(G) = \{v_0, v_1, ..., v_{n-1}\}$, is interval iff:

There is a *path* P and a collection $\{p_0, p_1, ..., p_{n-1}\}$ of *paths* in P, such that: $P = \bigcup_{i=0}^{n-1} p_i$ and $\forall i, j \in [0, n-1], v_i v_j \in E(G) \Leftrightarrow V(p_i) \cap V(p_j) \neq \emptyset$.

We call such a path and collection of subpaths a ***path-path model*** of G.

In [2] the sets of path-path models of interval graphs are characterized using a data structure called PQ-trees. In particular, a PQ-tree can be used to represent the set of path-path models of a given interval graph. Additionally, a PQ-tree can be constructed for a given graph G in $\mathcal{O}(n+m)$ time. The PQ-tree has also seen numerous applications. Some of these include linear time: isomorphism testing for interval graphs [16], recognition for interval graphs [2], and consecutive ones property testing the of a matrix [2].

The ***path graphs*** – intersection graphs of ***paths of a tree*** – are the primary focus of this work and are an intermediate class between interval graphs and chordal graphs. In particular, a graph G with $V(G) = \{v_0, v_1, ..., v_{n-1}\}$, is a path graph iff:

There exists a *tree* T and a collection $\{p_0, p_1, ..., p_{n-1}\}$ of *paths* in T, such that: $T = \bigcup_{i=0}^{n-1} p_i$ and $\forall i, j \in [0, n-1], v_i v_j \in E(G) \Leftrightarrow V(p_i) \cap V(p_j) \neq \emptyset$.

We call such a tree and collection of subpaths a ***path-tree model*** of G.

The intersection graphs of *directed paths in directed trees* (i.e., the ***directed path graphs***) are also of interest in this paper.

[1] More information on intersection graphs is available in many texts (i.e., [17]).

These graph classes are related by the following proper inclusions: interval $\subset$ directed path $\subset$ path $\subset$ chordal [18].

A lot of work has been done to characterize path graphs, directed path graphs, and other similar chordal graph classes. The *clique tree theorems* (presented in detail in section 2.1) by Gavril (path graphs [10], and chordal graphs [9]), Monma and Wei (on directed path graphs [18]) and Gilmore and Hoffman (on interval graphs [11]) tell us that we can greatly restrict the possibilities for the corresponding model of each of these graph classes, i.e., the model of a graph G must be a spanning subgraph of G's clique graph [2]. The clique tree theorems were leveraged by Monma and Wei to develop the *clique separator theorems* (see [18]), which characterize each of these graph classes by considering the local structure surrounding clique separators. There are also *forbidden induced subgraph characterizations* (FISC) of chordal graphs (by definition), interval graphs (by combining results from Lekkerkerker and Boland [14] and Köhler [12]), directed path graphs (by Panda [19]), and, path graphs (by Lèvêque et al. [15]).

The recognition problem for these graph classes has also been well studied and polynomial time recognition algorithms are known for all of these graph classes. The best known recognition algorithm for path graphs is by Schaffer [21] and runs in $\mathcal{O}(n * m)$ time (this algorithm follows the approach described by Monma and Wei [18] via their clique separator theorem). Also, a proposed linear time recognition algorithm for path graphs is outlined in an extended abstract (by Dahlhaus and Bailey [7]) but, according to Dahlhaus [6], this work will ***not*** appear in a journal submission. For directed path graphs, the complexity of recognition is known to be $\mathcal{O}(n^2 * m)$ via Monma and Wei's clique separator theorem [18]. The complexity of recognition for directed path graphs has recently been improved to $\mathcal{O}(n * m)$ by Chaplick et al. [4]. Additionally, interval graphs were first shown to be recognizable in linear time by Booth and Lueker [2] (using PQ-trees) and a simple linear time recognition algorithm has been developed by Corneil et al. [5]. Chordal graphs also have a simple linear time recognition algorithm due to Rose et al. [20].

In this paper, we introduce the PR-tree and strong PR-tree (see section 3) data structures to characterize the sets of path-tree models of path graphs and the directed path-tree models of directed path graphs respectively. We will see that the strong PR-tree is a slight strengthening of the definition of the PR-tree. Additionally, we demonstrate a degenerate form of the PR-tree which characterizes the sets of path-path models of interval graphs (see theorem 5.3). This degenerate form of the PR-tree is essentially the PQ-tree of Booth and Lueker [2]. We also present an algorithm to construct a PR-tree

[2] The vertex set of the clique graph of a graph G is $\mathcal{K}_G$ and the edge set is the pairs of maximal cliques whose intersection is not empty.

from a given graph G (see section 4). In particular, the PR-tree characterizes G by capturing all of its path-tree models. Also, we extend this algorithm to construct a strong PR-tree from a given graph. The size of any PR-tree or strong PR-tree produced by these algorithms is linear (i.e., $\mathcal{O}(n+m)$) with respect to the size of the input graph. Additionally, both the algorithm and its extended version have a running time of $\mathcal{O}(a(n+m)*n*m)$ (where $a(s)$ is the inverse of Ackermann's function arising from Union-Find as presented in the addendum [3]). This nearly matches the fastest known recognition algorithms for path graphs [21] and directed path graphs [4] (each of which has a runtime complexity of $\mathcal{O}(n*m)$).

We begin with some background results regarding these chordal graph classes (see section 2). In the next section we define the PR-tree and and strong PR-tree, and prove some key theorems which lead to their characterizations of path graphs and directed path graphs. This is followed by the algorithm to construct a PR-tree from a given graph (see section 4 - we also provide the modified version used for strong PR-trees). Sections 5[3] and 6 present the correctness and runtime complexity of our PR-tree construction algorithms. The paper ends with some concluding remarks and consequences of this work (see section 7).

2 Background

In the first subsection of this section we discuss the Clique Tree Theorem for the path, directed path, interval, and chordal graph classes. We then relate the Clique Tree Theorem to characterizing the respective sets of models of path graphs, directed path graphs, and interval graphs. The latter subsection provides the basis for our data structures (i.e., the PR-tree and strong PR-tree presented in section 3)) and construction algorithms (see section 4).

2.1 The Clique Tree Theorem

In the definition of the path graph class there are no restrictions regarding how we can choose the path-tree model which demonstrates a graph's membership in this class. In particular, any tree and collection of subpaths may be selected as long as they satisfy the intersection conditions. This is similarly true for the directed path, interval, and chordal graph classes. In this section we present a very useful restriction on the models of these graph classes in terms of their maximal cliques. This is known as ***the clique tree theorem***. In particular, the clique tree theorem tells us that a graph G belongs to the path / directed path / interval / chordal graph class iff a

[3] The degenerate PR-tree for interval graphs is also discussed in this section.

respective model can be constructed from a host graph H whose vertex set is the set of maximal cliques of G (i.e., $\mathcal{K}_G$) and the subset of $V(H)$ corresponding to a vertex $v \in V(G)$ is the set of maximal cliques incident with v (i.e., S_v). This is formalized in the following theorem:

Theorem 2.1.1 *(**Clique Tree**) A graph G =(V={v_0, v_1, ..., v_{n-1}},E) is:*

*(1) [10] A **path** graph iff there exists a **tree** T with vertex set $\mathcal{K}_G$, such that for every $i \in [0, n-1]$, T induced on S_{v_i} is a **path**.*

*(2) [18] A **directed path** graph iff there exists a **directed tree** T with vertex set $\mathcal{K}_G$, such that for every $i \in [0, n-1]$, T induced on S_{v_i} is a **directed path***

*(3) [11] An **interval** graph iff there exists a **path** P with vertex set $\mathcal{K}_G$, such that for every $i \in [0, n-1]$, P induced on S_{v_i} is a **path**.*

*(4) [9] A **chordal** graph iff there exists a **tree** T with vertex set $\mathcal{K}_G$, such that for every $i \in [0, n-1]$, T induced on S_{v_i} is a **tree**.*

Note: We use $\mathcal{T}_G$, $\overrightarrow{\mathcal{T}}_G$, and $\mathcal{P}_G$ to denote the set of all such path-tree, directed path-tree, and path-path models (as in (1), (2), and (3) respectively). For example:
$\mathcal{T}_G = \{T : T$ *is a **tree** with* $V(T) = \mathcal{K}_G$, *and* $\forall\ i \in [0, n-1]$, $T[S_{v_i}]$ *is a **path***$\}$.

From here on we consider all path-tree, directed path-tree, path-path, and tree-tree models to be the clique versions as outlined in the Clique Tree Theorem (i.e., a ***path-tree model*** of a graph G is a tree, T, with vertex set $\mathcal{K}_G$ such that, for every vertex v of G, S_v induces a path in T). The tree-tree model as we have presented here is more commonly referred to as a ***clique tree***. We have chosen the name tree-tree model so that it is easier to differentiate between a clique tree and the more restrictive versions of clique trees used by restricted subclasses of chordal graphs (i.e., the path, directed path, and interval graph classes). A detailed discussion of clique trees can be found in many graph theory texts (for example [17]).

2.2 The Path-Tree, Directed Path-Tree, and Path-Path Problems

Using the clique tree theorem we can now formalize what we mean by having a data structure which can characterize the set of path-tree / directed path-tree / path-path models associated with any given graph.

We use the path-tree models as our canonical case for establishing this concept. In particular, a data structure must be devised such that, for an arbitrary graph G, there is an instance of this data structure which captures the path-tree models of G in a "concise" manner. Furthermore, an algorithm must be devised which, for a given G, produces such an instance of the data structure. The PR-tree and its corresponding construction algorithm accomplish this goal. Additionally, the strong PR-tree and its construction algorithm accomplish this goal for directed path-tree models. Booth and Lueker [2] have already accomplished this for path-path models with the

PQ-tree and its corresponding construction algorithm. In observation 3.4.2 we relate a degenerate form of PR-trees to PQ-trees (note: this degenerate form is, for all intents and purposes, the same as a PQ-tree).

We begin to formalize the specification of each construction algorithm by defining the path-tree, directed path-tree, and path-path problems.

Definition 2.2.1 *Given a graph G the **path-tree problem** is to determine the set $\mathcal{T}_G$ of path-tree models of G, where $\mathcal{T}_G$ is captured in a "concise" manner. The **directed path-tree** and **path-path** problems are defined similarly using $\overrightarrow{\mathcal{T}}_G$ and $\mathcal{P}_G$ in place of $\mathcal{T}_G$ (respectively).*

Notice that, the set $\mathcal{T}_G$ is the subset of all trees over the vertex set $\mathcal{K}_G$ in which S_v is a path for all $v \in V(G)$. In this framing, each set of incident maximal cliques can be thought of as a ***constraint***. In particular, we define the ***constraint set*** of a graph G to be the set $\mathbb{S}_G = \{S_v : v \in V(G)\}$.

To explain what we mean by "concise", we apply a general approach (from [1]) to the problem of constructing $\mathcal{T}_G$ (see algorithm 1 below). This is called a ***Reduction*** because it iteratively reduces the set of all trees to those which satisfy the constraints.

Algorithm 1 ***Reduction($\mathbb{S}$), version 1:***
Determining the set $\mathcal{T}_G$ from $\mathbb{S}_G$ for a graph G by reduction [1].

```
1 #pre: $\mathbb{S} = \mathbb{S}_G$ is the constraint set of a graph $G$ (i.e, $\mathcal{K}_G = \bigcup_{S \in \mathbb{S}} S$).
2 #post: $\mathcal{T}$ is the set of path−tree models of $G$ (i.e., $\mathcal{T} = \mathcal{T}_G$).
3   $\mathcal{T} = \{T : T$ is a tree and V($T$) =$\mathcal{K}_G\}$  # all possible trees over $\mathcal{K}_G$.
4   for $S \in \mathbb{S}$:
5     $\mathcal{T} = \mathcal{T} \setminus \{T : T \in \mathcal{T}$, and $T[S]$ is not a path$\}$
6   return $\mathcal{T}$
```

If we were to implement this algorithm by representing $\mathcal{T}$ as a list, it would be quite easy, but also quite inefficient. The key to Algorithm 1's success relies entirely on having a data structure that represents the set $\mathcal{T}$ in a concise manner and allow the introduction of new constraints (as in lines 4-5 of Algorithm 1), to be performed efficiently. Furthermore, representing the set $\mathcal{T}_G$ as a list is not a good idea since this could be exponentially large with respect to G. The data structure that we use for this is the PR-tree.

3 PR-trees

The class of PR-trees over $\mathcal{U} = \{u_0, u_1, ..., u_{z-1}\}$ is defined to be all multi-sourced directed trees where: each leaf is an element of $\mathcal{U}$, each internal (non-leaf, non-source) node is a P-node or an R′-node, and each source node is a P-node, an R-node, or an R$'$-node [4]. Since a PR-tree is a multi-sourced directed tree, each non-source node could have more than one parent node and more than one source node. Thus for a given node N, we let $\boldsymbol{Parents(N)}$ be the parent nodes of N and $\boldsymbol{Sources(N)}$ be the source nodes of N (note: when N is a source node, $Sources(N) = \{N\}$). Similarly, we let $\boldsymbol{Children(N)}$ be the children of N and $\boldsymbol{L(N)}$ be the leaves of N (i.e., when N is a leaf $L(N) = \{N\}$). Also, we use $\boldsymbol{Desc(N)}$ to denote the descendents of N (i.e., when N is a leaf, $Desc(N) = \emptyset$).

In this section we formalize the PR-tree and prove that a PR-tree over the set $\mathcal{U}$ represents a set of trees over $\mathcal{U}$. Due to the conceptual complexity of the PR-tree we present its definition using a staged approach. In particular, we first present the ***weak PR-tree*** – a less restrictive version of the PR-tree. The weak PR-tree is a directed acyclic graph over a less restrictive set of nodes (P-nodes, ***weak*** R-nodes, and R$'$-nodes). This weaker structure not only provides a clearer understanding of the PR-tree, but it is also useful for our algorithm in section 4 as its properties are easier to maintain than those of the PR-tree. In subsection 3.2 we establish the PR-tree by restricting the weak PR-tree. Then, in subsection 3.3, we demonstrate that a PR-tree over $\mathcal{U}$ represents a tree over $\mathcal{U}$. In section 3.4 we define an equivalence class for PR-trees and present the relationship between PQ-trees and PR-trees. The final subsection (3.5) discusses least common ancestors (LCAs) in PR-trees and demonstrates a relationship between the sets of path-tree models of graphs and PR-trees with respect to least common ancestors. Throughout this section we also describe the ***strong PR-tree*** (the restricted PR-tree used for the directed path-tree problem).

3.1 The weak PR-tree

The purpose of this subsection is to introduce the ***weak PR-tree*** through its building blocks. We first define these building blocks (i.e., the nodes that make up a weak PR-tree): the ***P-node***, ***weak R-node***, and ***R$'$-node***.

We use the term ***weak R-node*** to separate the definition of the ***R-node*** into two important pieces. The first of which (def. 3.1.2 below) provides the overall structure of the R-node. The full definition of the R-node appears in subsection 3.2 (see def. 3.2.8) and describes additional conditions required to ensure the path-tree models are properly represented.

Also, even though the weak PR-tree is a directed acyclic graph, we insist that the rooted subgraph induced by each node is a rooted tree. This is enforced by following property 4 in def. 3.1.1, 3.1.2, 3.1.3.

Definition 3.1.1 *P is a **P-node** with* $Children(P) = \{N_0, N_1, ..., N_{k-1}\}$ *when:*

[4] An element of $\mathcal{U}$ can be a source node, but only when $|\mathcal{U}| = 1$.

(1) $k > 2$*; and*
(2) $\forall i \in [0, k-1]$*,* N_i *is an R′-node, a P-node, or an element of* $\mathcal{U}$*; and*
(3) The children of P are ordered $N_0 < N_1 < ... < N_{k-1}$*; and*
(4) No two children of P have leaves in common, i.e., $\forall i, j \in [0, k-1]$*,* $i \neq j \Rightarrow L(N_i) \cap L(N_j) = \emptyset$*.*

Note: We draw a P-node as a circle with its children drawn below it (see fig. 1).

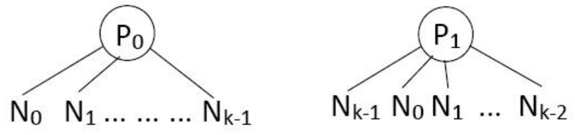

Fig. 1. Two P-nodes with the set of children $\{N_0, N_1, ..., N_{k-1}\}$ where P_0 has the order: $N_0 < N_1 < ... < N_{k-1}$ and P_1 has the order: $N_{k-1} < N_0 < N_1 <... < N_{k-2}$.

Notice that every P-node has at least three children. The reason for doing this is to avoid a potential ambiguity. In particular, as we will see later (see subsection 3.4), a P-node with two children is effectively the same as an R′-node with two children. This choice of disambiguation stems from the relationship between R′-nodes and R-nodes, which we discuss next.

Before formally defining weak R-nodes and R′-nodes, we first discuss the relationship between these nodes. For every weak R-node $\mathcal{R}$ there is a non-empty set of R′-nodes $\mathcal{R}_{R'}$ where $Children(\mathcal{R}) = \bigcup_{Q \in \mathcal{R}_{R'}} Children(Q)$. Similarly, for every R′-node Q there is a unique weak R-node Q_R such that every child of Q is a child of Q_R and no other weak R-node has children in common with Q. Also, the children of an R′-node Q are ordered (similarly to the children of a P-node) and this order is constrained by Q's weak R-node.

Definition 3.1.2 $\mathcal{R} = \{\mathcal{R}_T, \mathcal{R}_{R'}\}$ *is a* ***weak R-node*** *with* $Children(\mathcal{R}) = \{N_0, N_1, ..., N_{k-1}\}$ *when:*
(1) $k > 1$*; and*
(2) $\boldsymbol{\mathcal{R}_T}$ *is a tree whose nodes are the children of* $\mathcal{R}$*; and*
(3) $\boldsymbol{\mathcal{R}_{R'}}$ *is the set of all R′-nodes which are parents of* $\mathcal{R}$*'s children (i.e.,* $\mathcal{R}_{R'} = \{Q : \exists i \in [0, k-1]$*,* $Q \in$ *Parents*(N_i) *and Q is an R′-node*$\}$*); and*
(4) No two children of $\mathcal{R}$ *have leaves in common; and*
(5) $\forall i \in [0, k-1]$*,* N_i *is an R′-node, a P-node, or an element of* $\mathcal{U}$*, where:*
  (a) No other weak R-node may have children in common with $\mathcal{R}$ *(i.e.,* $\forall N \in (Parents(N_i) - \{\mathcal{R}\})$*, N is a P-node or an R′-node); and*
  (b) If $N_i N_j$ *(*$j \in [0, k-1]$*) is an edge in* $\mathcal{R}_T$*, then there is an R′-node* $Q \in \mathcal{R}_{R'}$ *such that* N_i *and* N_j *are children of Q.*

Note: To visualize a weak R-node $\mathcal{R}$ *we use* $\mathcal{R}_{R'}$ *and* $\mathcal{R}_T$ *(see fig. 2 below).*

Definition 3.1.3 *Q is an* ***R′-node*** *with* $Children(Q) = \{N_0, N_1, ..., N_{k-1}\}$ *if:*
(1) $k > 1$*; and*
(2) $\forall i \in [0, k-1]$*,* N_i *is an R′-node, a P-node, or an element of* $\mathcal{U}$*; and*
(3) The children of Q are ordered $N_0 < N_1 < ... < N_{k-1}$*; and*

(4) *No two children of Q have leaves in common; and*

(5) *There exists a weak R-node Q_R with tree $(Q_R)_T$, such that $\forall i \in [0, k-1], Q_R \in Parents(N_i)$ and N_0, N_1, ..., N_{k-1} is a path in $(Q_R)_T$. Note: from property 5a of weak R-nodes (def. 3.1.2 above), Q_R is the only weak R-node with children in common with Q, thus Q_R is **the** weak R-node of Q.*

Note: We draw an R′-node as a rectangle with its children below it (see fig. 2).

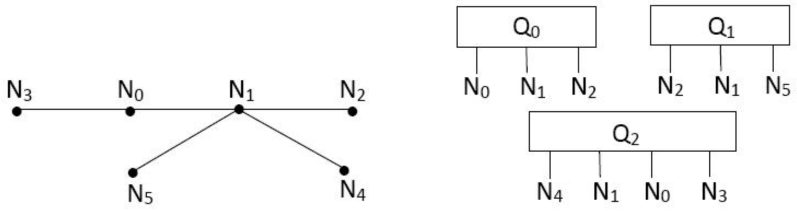

Fig. 2. Weak R-node with tree $\mathcal{R}_T$ (left) and R′-nodes $\mathcal{R}_{R'} = \{Q_0, Q_1, Q_2\}$ (right). Note: each R′-node orders its children as shown, i.e., Q_2 has $N_4 < N_1 < N_0 < N_3$.

We now provide a couple of observations to emphasize the relationship between weak R-nodes and R′-nodes.

Observation 3.1.4 *If $\mathcal{R}$ is a weak R-node, Q is an R′-node, and $Children(Q) \cap Children(\mathcal{R}) \neq \emptyset$, then $Children(Q) \subseteq Children(\mathcal{R})$, $Q_R = \mathcal{R}$ and $Q \in \mathcal{R}_{R'}$.*

Observation 3.1.5 *If Q_0 and Q_1 are R′-nodes and $Children(Q_0) \cap Children(Q_1) \neq \emptyset$, then $(Q_0)_R = (Q_1)_R$.*

Notice that property 4 of def. 3.1.1, 3.1.2, and 3.1.3 implies that no pair of overlapping nodes has a common source (see observation 3.1.6).

Observation 3.1.6 *Let N_1 and N_2 be nodes contained in a weak PR-tree.*
If $L(N_1) \cap L(N_2) \neq \emptyset$, $N_1 \notin Desc(N_2)$, and $N_2 \notin Desc(N_1)$, then:
Sources(N_1) ∩ Sources(N_2) = ∅.

We can now define the ***weak PR-tree***. This "weaker" version of the PR-tree is be extremely useful for our PR-tree construction algorithm in section 4. It is important to note that the weak R-nodes have disjoint children and partition the R′-nodes into disjoint sets (by definition).

Definition 3.1.7 *$D = \{\{A_0, ..., A_{\alpha-1}\}, \{R_0, ..., R_{\gamma-1}\}\}$ is a **weak PR-tree** over a set $\mathcal{U}$ when:*

(1) *D is a multi-sourced directed acyclic graph with source nodes: A_0, A_1, ..., $A_{\alpha-1}$, R_0, R_1, ..., $R_{\gamma-1}$; where:*

(2) *$\{A_0, A_1, ..., A_{\alpha-1}\}$ are P-nodes and R′-nodes [5]; and*

(3) *$\{R_0, R_1, ..., R_{\gamma-1}\}$ are weak R-nodes; and*

(4) *The leaf set of D is $\mathcal{U}$.*

[5] A source node can be an element of $\mathcal{U}$, but this only occurs in the degenerate case when $|\mathcal{U}| = 1$, i.e., when D is the single leaf u $\in \mathcal{U}$.

Note: From the definitions of P-nodes, weak R-nodes, and R′-nodes the internal nodes of D are P-nodes and R′-nodes. Also, to visualize the information contained in a PR-tree, we draw each ordered tree rooted at A_i, for $i \in [0, \alpha - 1]$, and each $(\mathcal{R}_j)_T$, for $j \in [0, \gamma - 1]$ (see fig. 3).

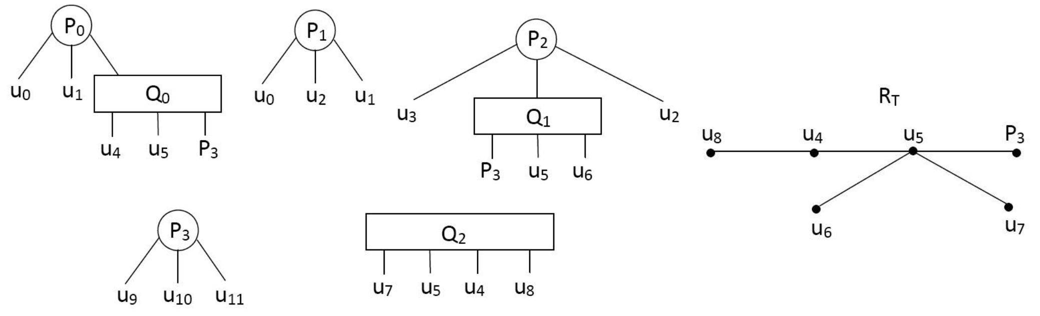

Fig. 3. A weak PR-tree D = $\{\{P_0, P_1, P_2, Q_2\}, \{R\}\}$, where $R_{R'} = \{Q_0, Q_1, Q_2\}$. Note: we draw nodes with multiple parents separately as with the node P_3.

To provide some insight into the weak PR-tree, we examine the result of re-ordering the children of P-nodes and R′-nodes in a given weak PR-tree. By considering the two P-nodes in fig. 1 (on page 8), it is clear that we can arbitrarily permute the children of a P-node in a weak PR-tree and the result is a new P-node (we also get a new weak PR-tree). However, if we attempt to apply the same permutation to the R′-node, Q_0 (as in fig. 2), the result would not be a valid R′-node because the new order would not represent a path in $(Q_0)_R$. Thus, there is only one re-ordering of the children of Q that results in a new R′-node. In particular, the only valid re-ordering is to reverse the order imposed by Q (i.e., if the original order on $Children(Q)$ = $\{N_0, N_1, ..., N_{k-1}\}$ was $N_0 < N_1 < ... < N_{k-1}$, then the new order would be $N_{k-1} < N_{k-2} < ... < N_0$ and is still a path in Q_R). We formalize these operations as ***equivalence transformations*** in subsection 3.4.

In the next section, we examine the goal of the PR-tree (i.e., representing path-tree models) to show why the weak PR-tree is not strict enough.

3.2 *Formalizing PR-trees*

The purpose of this section is to formalize the PR-tree. In particular, we will first demonstrate the two "weaknesses" of a weak PR-tree to motivate the PR-tree's more restrictive definition. Then we present this definition.

More specifically, a given PR-tree D will directly map to a tree (whose vertices are the leaves of D) and we will call this tree the ***frontier*** of D. The other trees that D represents correspond to PR-trees ***equivalent*** to D (see section 3.4). The frontiers of weak PR-trees and weak R-nodes are stated in terms of the frontiers of P-nodes and R′-nodes where the frontiers of P-nodes and R′-nodes correspond to the leaf order they specify (see observation 3.2.1 and def. 3.2.2). The frontier of a weak PR-tree will drive the derivation of the definition of the PR-tree.

Observation 3.2.1 *Since P-nodes and R′-nodes are ordered and their children are P-nodes, R′-nodes, or leaves (which are also ordered), thus a P-node or R′-node N specifies an ordered subtree of its weak PR-tree, and, consequently, an ordering* $\boldsymbol{\sigma(N)} = u_0 < u_1 < ... < u_{z-1}$ *on its leaf set* $L(N) = \{u_0, u_1, ..., u_{z-1}\}$. *Note: if N is an element of $\mathcal{U}$ then $L(N) = \{N\}$ and is trivially ordered.*

Definition 3.2.2 *We use* $\boldsymbol{Frontier(X)}$ *to denote the frontier of X, where the **frontier** of a:*
- *P-node, R′-node or leaf node N is the graph consisting of the path $\sigma(N)$.*
- *weak R-node $\mathcal{R}$ is the graph:* $\bigcup_{Q \in \mathcal{R}_{R'}} Frontier(Q)$.
- *weak PR-tree* $D = \{\{A_0, ..., A_{\alpha-1}\}, \{R_0, ..., R_{\gamma-1}\}\}$ *is the graph* $\bigcup_{i=0}^{\alpha-1} Frontier(A_i)$.

Fig. 4 and 5 show the frontiers of a weak R-node and weak PR-tree resp.

Notice that, we can form a directed analogue of the frontier (i.e., the ***directed frontier***) by using $\sigma(N)$ as a directed path. In particular, we have the following definition which is useful for the directed path tree problem.

Definition 3.2.3 *We use* $\boldsymbol{\overrightarrow{Frontier}(X)}$ *to denote the directed frontier of X, where the **directed frontier** of a:*
- *P-node, R′-node or leaf node N is the directed path $\sigma(N)$.*
- *weak R-node $\mathcal{R}$ is the directed graph:* $\bigcup_{Q \in \mathcal{R}_{R'}} \overrightarrow{Frontier}(Q)$.
- *weak PR-tree* D $=\{\{A_0, ..., A_{\alpha-1}\}, \{R_0, ..., R_{\gamma-1}\}\}$ *is the directed graph:* $\bigcup_{i=0}^{\alpha-1} \overrightarrow{Frontier}(A_i)$.

Fig. 4 and 5 show the directed frontiers of a weak R-node and weak PR-tree resp.

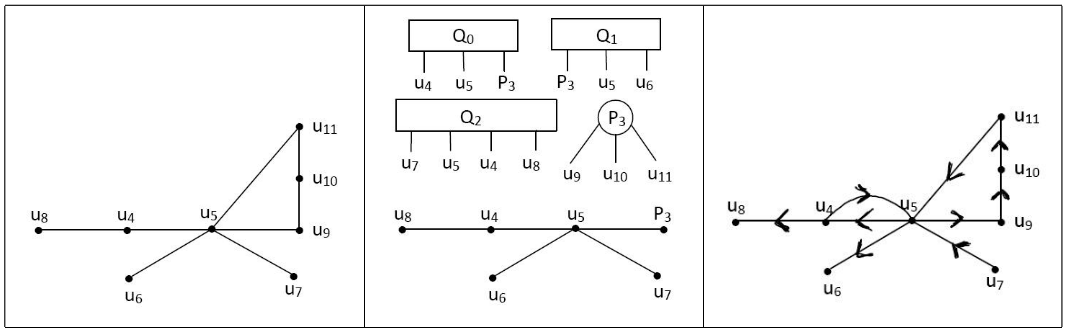

Fig. 4. The weak R-node R from fig. 3 (centre), R's frontier (left), and R's directed frontier (right).

Observation 3.2.4 *Since the frontier of a weak R-node $\mathcal{R}$ in a weak PR-tree D is the union of the frontiers of $\mathcal{R}$'s R′-nodes (each of which is either a source node or a descendent of a source node with respect to D), therefore,* $\boldsymbol{Frontier(\mathcal{R})}$ ***is a subgraph of*** $\boldsymbol{Frontier(D)}$. *Note: This is why we do not need to explicitly include the frontier of weak R-nodes in the frontier of weak PR-trees.*

Notice that, for a given weak R-node $\mathcal{R} = \{\mathcal{R}_T, \mathcal{R}_{R'}\}$ in a weak PR-tree D we can create a weak PR-tree $D_{\mathcal{R}}$ where $Frontier(D_{\mathcal{R}})$ = $Frontier(\mathcal{R})$ (i.e., we

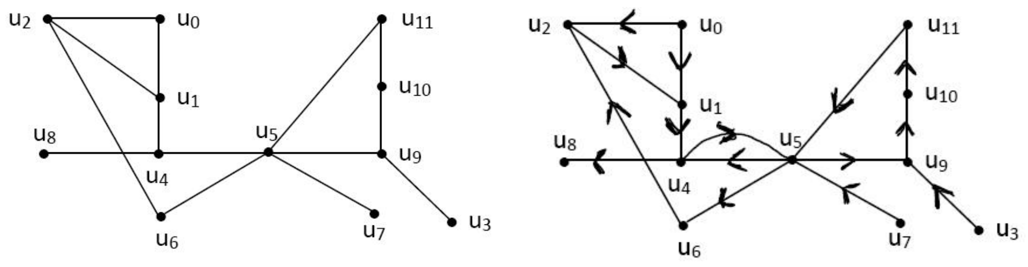

Fig. 5. The frontier (left) and directed frontier (right) of the weak PR-tree in fig. 3.

use the set $\mathcal{R}_{R'}$ as the source nodes of D_R that are not weak R-nodes). For example, for the weak R-node R in fig. 4, we have D_R = $\{\{Q_0, Q_1, Q_2\}, \{\text{R}\}\}$ and it is drawn identically to R (i.e., as in fig. 4 on page 11).

Observation 3.2.5 *If D is a weak PR-tree and $\mathcal{R}$ is a weak R-node in D then $Frontier(D_{\mathcal{R}})$ = $Frontier(\mathcal{R})$. This follows from def. 3.2.2 (i.e., of frontier).*

We now demonstrate the two shortcomings of the weak PR-tree. Notice that the weak R-node in fig. 4 (on page 11) has a cycle in its frontier. Therefore, since the frontier of every R-node must be a tree, we need a stricter definition for R-nodes. In particular, when two R′-nodes have two children in common (i.e., Q_0 and Q_1 from fig. 4) and do not enforce the "same" ordering on those children (i.e., Q_0 has $u_5 < P_3$ and Q_1 has $P_3 < u_5$), then the frontier of their corresponding weak R-node will contain a cycle. Additionally, the directed frontier of that same weak R-node has a directed cycle u_4, u_5. This cycle is due to the R′-nodes Q_0 and Q_2 inconsistently ordering their children. However, in the frontier this pair does not cause a problem. Moreover, this differentiates the conditions needed for the undirected and directed cases with the directed case being more restrictive (these are formalized in def. 3.2.6 and 3.2.7 respectively).

Definition 3.2.6 *A weak R-node $\mathcal{R}$ is **pairwise well-formed (PWF)** when:*
*There is **no** pair $\{N_0, N_1\} \subseteq \mathcal{R}_{R'}$ such that:*
N_0 and N_1 have at least two common children X_0 and X_1 where (w.l.o.g.):
- *$X_0 < X_1$ with respect to N_0 and $X_1 < X_0$ with respect to N_1; and*
- *X_0 has more than one leaf.*

Definition 3.2.7 *A weak R-node $\mathcal{R}$ is **pairwise consistent (PC)** when:*
*There is **no** pair $\{N_0, N_1\} \subseteq \mathcal{R}_{R'}$ such that:*
N_0 and N_1 have at least two common children X_0 and X_1 where (w.l.o.g.):
- *$X_0 < X_1$ with respect to N_0 and $X_1 < X_0$ with respect to N_1.*

Definition 3.2.8 *The **R-nodes** are the PWF weak R-nodes (def. 3.2.6 and 3.1.2). Similarly, the **strong R-nodes** are the PC weak R-nodes (def. 3.2.7 and 3.1.2). Note: Both are drawn identically to weak R-nodes. See fig. 2 on page 9.*

Consider the weak PR-tree from fig. 3 on page 10. Notice that, if we reverse the order specified by Q_1 on its children and call this new R′-node Q'_1, then

Q'_1 will represent the same path as Q_1 did in $((Q_1)_R)_T)$ and we will have a weak PR-tree where every weak R-node is an R-node (see fig. 6 below).

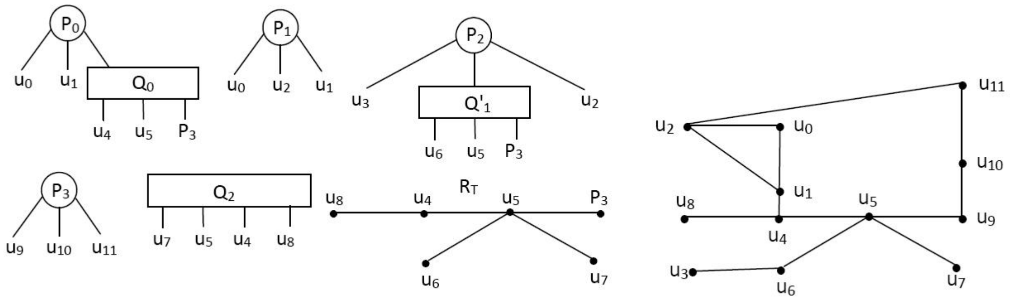

Fig. 6. Revised version of the weak PR-tree from fig. 3 where the weak R-node is now an R-node (left) and its frontier (right).

Since the frontier of the weak PR-tree in fig. 6 (above) is not acyclic, there is another restriction that must be applied to the weak PR-tree in order to guarantee that the frontier is acyclic. Consider the sequence of nodes P_0, P_2, P_1 from the weak PR-tree in fig. 6. Notice that, Adjacent pairs (i.e., (P_0, P_2), (P_2, P_1)) in the sequence have at least one common leaf (i.e., they overlap). The first and last nodes in the sequence also overlap.

We formalize this structure as a ***loop*** (def. 3.2.9 below), but we first examine the weak PR-tree in fig. 6 in more detail. Notice that there is no leaf shared among this entire sequence, and, consequently, there must be a cycle in the frontier of the weak PR-tree (i.e., the cycle: u_1, u_4, u_5, u_9, u_{10}, u_{11}, u_2). We can also have a loop (i.e., P_0, P_1) with shared leaves (i.e., u_0 and u_1) that results in a cycle (i.e., u_0, u_1, u_2) in the frontier. This cycle results from the fact that there is no "descendent" of both P_0 and P_1 whose leaf set is exactly the set of shared leaves. In particular, the absence of such a common "descendent" means that it is possible for the union of their frontiers to contain a cycle (i.e., $Frontier(P_0) \cup Frontier(P_1)$ contains a cycle). Therefore, to help ensure that the frontier of a loop is a tree, we will not only insist that the intersection of their leaf sets is not empty, but we also insist that there is a common "descendent" among all of the nodes in the loop. We refer to this as the ***helly property*** of weak PR-trees (see def. 3.2.10).

Definition 3.2.9 *In a given weak PR-tree D an ordered set X =* $\{N_0, N_1, ..., N_{k-1}\}$ *(*$k > 1$*) of nodes contained in D is a* ***loop*** *when:*
$\forall i, j \in [0, k-1]$, $N_i \notin Desc(N_j)$; *and* $\forall i \in [0, k-1]$, $L(N_i) \cap L(N_{i+1 (\text{mod k})}) \neq \emptyset$.

Note: We use $\boldsymbol{X_L}$ *to denote the set of leaves in common among every node in the loop, i.e.,* $X_L = \bigcap_{i=0}^{k-1} L(N_i)$.

Definition 3.2.10 *A weak PR-tree D is* ***helly*** *when every loop X =* $\{N_0, N_1, ..., N_{k-1}\}$ *in D has* $\boldsymbol{X_L \neq \emptyset}$ *and* ***either****:*
(1) There exists a node B in D where L(B) = X_L *and for every* $i \in [0, k-1]$*, either B =* N_i *or* $B \in Desc(N_i)$*. For example, the helly loop* $\{P_0, P_2, Q_2\}$ *(from fig.*

6 on page 13) would have B = u_5.

Note: In this case we use $\boldsymbol{X_B}$ to denote B.

OR

(2) There exist nodes C = $\{C_0, C_1, ..., C_{q-1}\}$ (q > 1) in D where $\bigcup_{j=0}^{q-1} L(C_i)$ = X_L and for every $i \in [0, k-1]$, there is an R′-node B_i where B_i = N_i or $B_i \in Desc(N_i)$ and C $\subseteq Children(B_i)$.

Note: Since the B_i's are overlapping R′-nodes, (by observation 3.1.5) we must have a weak R-node $\mathcal{R}$ = $\{\mathcal{R}_T, \mathcal{R}_{R'}\}$ where $\mathcal{R}$ is the weak R-node for every B_i (i.e., $\mathcal{R}$ = $(B_0)_R$ = $(B_1)_R$ = ... = $(B_{k-1})_R$). Furthermore, C_0, C_1, ..., C_{q-1} specifies a path P in $\mathcal{R}_T$. Thus, when property (1) is not satisfied we use $\boldsymbol{X_B}$ to denote an R′-node corresponding to P (i.e., $Children(X_B)$ = $\{C_0, C_1, ..., C_{q-1}\}$ and X_B orders C_0, C_1, ..., C_{q-1} according to some B_i[6]). Also, it is possible that X_B itself does not exist in D. For example, the helly loop $\{P_0, P_2\}$ (from the weak PR-tree in fig. 6) would have B_0 = Q_0 and B_1 = Q'_1, and X_B would be an R′-node with $Children(X_B)$ = $\{u_5, P_3\}$ and $u_5 < P_3$, but X_B is not an explicit R′-node in the weak PR-tree.

Note: X_L = $L(X_B)$.

We define the PR-tree and strong PR-tree as follows (see def. 3.2.11 below).

Definition 3.2.11 *D = $\{\{A_0, ..., A_{\alpha-1}\}, \{R_0, ..., R_{\gamma-1}\}\}$ is a:*

- ***PR-tree*** *over a set $\mathcal{U}$ when D is a weak PR-tree (def. 3.1.7) over $\mathcal{U}$, D is helly (def. 3.2.10) and every weak R-node in D is an R-node (def. 3.2.8).*
- ***strong PR-tree*** *over a set $\mathcal{U}$ when D is a weak PR-tree (def. 3.1.7) over $\mathcal{U}$, D is helly (def. 3.2.10) and every weak R-node in D is a strong R-node (def. 3.2.8).*

Note: From the definitions of P-nodes, weak R-nodes, and R′-nodes the internal nodes of D are P-nodes and R′-nodes.

In the next two subsections, we prove that the frontier of a PR-tree is a tree and demonstrate an equivalence class on PR-trees that shows how a PR-tree captures the path-tree models of a graph.

3.3 The frontier of a PR-tree is a Tree

The next step is to show that the frontier of a PR-tree is a tree. To do this we first note that the frontier of a PR-tree is connected (see observation 3.3.1). The key lemma of this section is then proven (see lemma 3.3.2). In particular, we prove that every loop in a PR-tree has a common subpath among the frontiers of all of its nodes. Using this result we then prove that if D is a helly weak PR-tree (***HW-PR-tree***) where every weak R-node is an R-node (i.e., D is a PR-tree) then the frontier of D is a tree. Finally we demonstrate that the frontier of a weak R-node is acyclic iff it is an R-node.

[6] Although either orderings on C (i.e., w.l.o.g. $C_0 < ... < C_{q-1}$ or $C_{q-1} < ... < C_0$) satisfy the tree $\mathcal{R}_T$, we insist that X_B uses an order imposed by some B_i.

Observation 3.3.1 *The frontier of every weak PR-tree is connected.*

Proof:
Consider a weak PR-tree D = $\{\{A_0, ..., A_{\alpha-1}\}, \{R_0, ..., R_{\gamma-1}\}\}$. First, we notice that $Frontier(A_0)$ is connected (since it is a path). Now, (w.l.o.g.) consider $\{A_0, A_1, ..., A_{k-1}\}$ where $k \in [1, \alpha - 1]$ and $G_k = \bigcup_{i=0}^{k-1} Frontier(A_i)$ is connected. Since D is connected, we must have some $j \in [0, k-1]$ such that (w.l.o.g.) $L(A_k) \cap L(A_j) \neq \emptyset$ (i.e., $G_{k+1} = G_k \cup Frontier(A_k)$ is connected). Therefore, by induction $Frontier$(D) is connected.
QED.

Lemma 3.3.2 *If $X = \{N_0, N_1, ..., N_{k-1}\}$ is a loop in a PR-tree D then $Frontier(X_B)$ specifies a path P whose vertex set is X_L, such that for all $i \in [0, k-1]$, P is a subpath of $Frontier(N_i)$.*

Proof:
Since X is a loop in a PR-tree it must satisfy the helly property (def. 3.2.10). This provides us with two cases:

Case 1: X satisfies property (1) of the helly property.
Now, for every node N_i in X: either $X_B = N_i$ or $X_B \in Desc(N_i)$ (by property (1)). Therefore, $Frontier(X_B)$ specifies a path P which is a subpath of $Frontier(N_i)$ by the definition of frontier (see def. 3.2.2).

Case 2: X satisfies property (2) of the helly property.
Now, $\forall$ i $\in [0, k-1]$, let B_i be the R′-node descendent of N_i where $Children(X_B) \subseteq Children(B_i)$ and let P_i specify the path in $\mathcal{R}_T$ corresponding to B_i. Furthermore, let $\mathcal{R} = \{\mathcal{R}_T, \mathcal{R}_{R'}\}$ be the R-node, where $\mathcal{R} = (X_B)_R = (B_0)_R = (B_1)_R = ... = (B_{k-1})_R$. Finally, we let $Children(X_B) = \{C_0, C_1, ..., C_{q-1}\}$ ($q > 1$) and assume (w.l.o.g.) that X_B orders its children as follows: $C_0 < C_1 < ... < C_{q-1}$.

Then, P = $C_0, C_1, ..., C_{q-1}$ must be a path in $\mathcal{R}_T$, and $\forall\ i \in [0, k-1]$ P is a subpath of P_i, such that either:
(*) B_i orders $Children(X_B)$: $C_0 < C_1 < ... < C_{q-1}$; or
(**) B_i orders $Children(X_B)$: $C_{q-1} < C_{q-2} < ... < C_0$.
Note: These are the only possible orders since $C_0, C_1, ..., C_{q-1}$ specifies a path in $\mathcal{R}_T$. In particular, we know:
(***) $\exists\ j \in [0, k-1]$, where B_j orders $Children(X_B)$: $C_0 < ... < C_{q-1}$.
Now we can further subdivide this case into two more cases:

Case 2(a): $\forall$ i $\in [0, q-1]$, $|L(C_i)| = 1$.
In this case, every child of X_B is a leaf. Thus, not only is P a subpath of every P_i, but is it also a subpath of every $Frontier(B_i)$ and therefore $Frontier(N_i)$ by the definition of frontier (see def. 3.2.2).

Case 2(b): $\exists\ i \in [0, q-1]$, where $|L(C_i)| > 1$.
Suppose (for a contradiction): $\exists$ j* $\in [0, k-1]$, where B_{j^*} orders $Children(X_B)$: $C_{q-1} < C_{q-2} < ... < C_0$ (as in (**) above). Notice that $j^* \neq j$ (for j as in (***) above). But now, since q > 1, $\exists\ i^* \in [0, q-1]$, $i^* \neq i$, and (w.l.o.g.) $C_i < C_{i^*}$ with respect to B_j and $C_{i^*} < C_i$ with respect to B_{j^*} (this con-

tradicts the fact that $\mathcal{R}$ is PWF). Therefore, $\forall\ i \in [0, k-1]$ B_i orders $Children(X_B)$ the same as X_B (as in (*) above).

Therefore, $\forall\ i \in [0, k-1]$ $Frontier(X_B)$ is a subpath of $Frontier(B_i)$ and $Frontier(N_i)$ by the definition of frontier (see def. 3.2.2).

QED.

Theorem 3.3.3 *If D is a PR-tree then $Frontier(D)$ is acyclic.*

Proof:

Let D = $\{\{A_0, ..., A_{\alpha-1}\}, \{R_0, ..., R_{\gamma-1}\}\}$. For i $\in$ [1, α], let $G_i = \bigcup_{j=0}^{i-1} Frontier(A_j)$. We prove "$G_i$ is acyclic" by induction on i $\in$ [1, α]. Note: $G_\alpha = Frontier(\mathrm{D})$.

In the base case we have G_1 = $Frontier(A_0)$ which is acyclic (since it is simply a path). We now assume that G_i is acyclic for a given i $\in$ [1, α-1] and show that G_{i+1} is also acyclic. Notice that G_{i+1} = $G_i \cup Frontier(A_i)$.

Suppose (for a contradiction) that G_{i+1} contains a cycle. Then, $Frontier(A_i)$ contains a subpath X = x_0, x_1, ..., x_{k-1} (for some k>1), where X is edge disjoint from a component C of G_i, and shares exactly vertices x_0 and x_{k-1} with C. Also, in C, there is a path Y = y_0, y_1, ..., y_{w-1} (for some w>1), where y_0 = x_0 and y_{w-1} = x_{k-1}.

Let F = $\{A_0^*, A_1^*, ..., A_{z-1}^*\} \subseteq \{A_0, A_1, ..., A_{i-1}\}$ such that for every edge $y_j\ y_{j+1}$ ($j \in [0, k-2]$) in Y, there exists some A^* in F where $Frontier(A^*)$ contains $y_j y_{j+1}$. Furthermore, (w.l.o.g.) we assume F is ordered such that $x_0 \in L(A_0^*)$, $x_{k-1} \in L(A_{z-1}^*)$, and for all h $\in$ [0, z-2], $L(A_h^*) \cap L(A_{h+1}^*) \neq \emptyset$.

Let F* = F $\cup$ $\{A_i\}$. Clearly F* is a loop and (since D is helly) it must have a non-empty common set of leaves L = $L(A_i) \cap (\bigcap_{j=0}^{z-1} L(A_j^*))$ = $\{u_0, u_1, ..., u_{q-1}\}$. Furthermore, (by lemma 3.3.2) there must be a path P = u_0, u_1, ..., u_{q-1} such that for every A in F*, P is a subpath of $Frontier(A)$ [7].

Also, (w.l.o.g. regarding the choice of which end of X is labelled x_0 and which end is labelled x_{k-1}) let K be the x_0 to u_0 subpath of $Frontier(A_{i+1})$ such that K includes x_{k-1}.

Notice that $A_0^* \in$ F and contains x_0. Now, $\{x_0, u_0\} \subseteq L(A_0^*) \cap L(A_i)$, thus $\{A_0^*, A_{i+1}\}$ is a loop. Then, (by lemma 3.3.2) $Frontier(A_{i+1})$ and $Frontier(A_0^*)$ must have the same subpath on $L(A_i) \cap L(A_0^*) \supseteq \{x_0, u_0\}$. Thus K is a subpath of C. However, K contains X as a subpath contradicting X being edge disjoint from C.

Therefore, G_{i+1} is acyclic and by induction $Frontier(\mathrm{D})$ = G_α is acyclic.

QED.

We now prove the following lemma to justify our definition of the R-node.

Lemma 3.3.4 *Let R be a weak R-node in a weak PR-tree D. Then $Frontier(R)$ is*

[7] Note: the path P must be contained in G_i and thus no u_i can be contained in X.

acyclic iff R is an R-node.

Proof:
($\Longrightarrow$) Suppose $Frontier(R)$ is acyclic and R is not PWF. Then there exist $\{N_0, N_1\} \subseteq R_{R'}$ and $\{X, Y\} \subseteq Children(N_0) \cap Children(N_1)$ with L(X) = $\{x_0, x_1, ..., x_{a-1}\}$ and L(Y) = $\{y_0, y_1, ..., y_{b-1}\}$ such that (w.l.o.g.): $a > 1$, $X < Y$ with respect to N_0, and $Y < X$ with respect to N_1.

Then $Frontier(N_0)$ contains the path $x_0, x_1, ..., x_{a-1}$, ..., $y_0, y_1, ..., y_{b-1}$, and $Frontier(N_1)$ contains the path $y_0, y_1, ..., y_{b-1}$, ..., $x_0, x_1, ..., x_{a-1}$. Therefore $Frontier(R)$ contains the cycle: $x_0, x_1, ..., x_{a-1}$, ..., $y_0, y_1, ..., y_{b-1}$, x_0.

Therefore, we have a contradiction.

($\Longleftarrow$) Suppose R is an R-node. Consider the weak PR-tree D_R. Then $Frontier(R) = Frontier(D_R)$ (by observation 3.2.5).

We claim that D_R is a PR-tree (i.e., its frontier is acyclic by theorem 3.3.3). Notice that every weak R-node in D_R is an R-node (since R is an R-node, and all other weak R-nodes are trivial). Thus, we just need D_R to be helly.

Now, by property 4 of weak R-nodes (see def. 3.1.2), if two nodes N_1 and N_2 in D_R have leaves in common (i.e., $L(N_1) \cap L(N_2) \neq \emptyset$) then:
(1) N_1 and N_2 must be elements of $R_{R'}$; and
(2) N_1 and N_2 must have at least one child in common (i.e., $Children(N_1) \cap Children(N_2) \neq \emptyset$).

Consider a loop X = $\{N_0, N_1, ..., N_{k-1}\}$ in D_R (i.e., $\forall\ i \in [0, k-1]$, $L(N_i) \cap L(N_{(i+1)\text{mod k}}) \neq \emptyset$, and $X_L = \bigcap_{i=0}^{k-1} L(N_i)$). Then, by (1) and (2) above:
X $\subseteq R_{R'}$, and $\forall\ i \in [0, k-1]$, $Children(N_i) \cap Children(N_{(i+1)\text{mod k}}) \neq \emptyset$.

Notice that, since each $N_i \in$ X corresponds to a path in R_T, we must have C = $\bigcap_{i=0}^{k-1} Children(N_i) \neq \emptyset$ (otherwise R_T would have to contain a cycle to satisfy the orders corresponding to every N_i). Therefore, $X_L = \bigcup_{c \in C}$ L(c) $\neq \emptyset$, and $\forall\ i \in [0, k-1]$ C $\subseteq Children(N_i)$ (i.e., X satisfies property 2 of the helly property (see def. 3.2.10)). Therefore, D_R is helly.
QED.

By the same techniques, a corresponding sequence of results follows for strong PR-trees and strong R-nodes (as in the below theorem).

Theorem 3.3.5 *Note: the proofs of (1), (2), and (3) follow similarly to those of lemma 3.3.2, theorem 3.3.3, and lemma 3.3.4 respectively.*
(1) If X = $\{N_0, N_1, ..., N_{k-1}\}$ is a loop in a strong PR-tree D then $Frontier(X_B)$ specifies a directed path P whose vertex set is X_L, such that for all $i \in [0, k-1]$, P is a directed subpath of $Frontier(N_i)$.
(2) The directed frontier of a strong PR-tree is acyclic.
(3) If R is a weak R-node in a weak PR-tree, then $\overrightarrow{Frontier}(R)$ is acyclic iff R is a strong R-node.

We have demonstrated that the frontier of every PR-tree is a tree over the leaves of D. We have also shown that every weak R-node must be an R-node in a weak PR-tree with an acyclic frontier. In the next section we describe an equivalence class for PR-trees.

3.4 PR-tree Equivalence

With the PR-tree and frontier of a PR-tree formally defined, we now describe the set of path-tree models represented by a PR-tree. As we have seen in section 3.3, a given PR-tree D directly maps to a tree (i.e., the frontier of D). The other elements in the solution set correspond to certain PR-trees ***equivalent*** to D (i.e., PR-trees that are the result of "re-ordering" the children of nodes within D).

As we discussed in subsection 3.1, an ***equivalence transformation*** of a P-node or R′-node N corresponds to a "valid re-ordering" of the children of N. In particular, since the children of an R′-node Q must form a path in $(Q_R)_T$, the only "valid re-ordering" on the children of Q would be to reverse their order. We use $\eta^{reverse}$ to denote this special transformation (i.e., for an R′-node Q with $Children(Q) = \{Q_0, Q_1, ..., Q_{k-1}\}$ in this order, $\eta^{reverse}(Q)$ is the R′-node with $Children(\eta^{reverse}(Q)) = Children(Q)$, and the children of $\eta^{reverse}(Q)$ are ordered $N_{k-1} < N_{k-2} < ... < N_0$). However, in the case of a P-node P we do not have any restrictions on the order P imposes on its children. Thus, any permutation of children of P is a "valid re-ordering." Similarly, for a weak PR-tree D an equivalence transformation corresponds to applying a "valid re-ordering" to the children of every P-node and R′-node contained in D (note: the identity re-ordering is also a "valid re-ordering"). We say that two weak PR-trees are ***equivalent*** when there exists one can be transformed into the other by applying an equivalence transformation.

Notice that, by applying an equivalence transformation to a weak PR-tree we can "change" its frontier. Thus, for a given weak PR-tree D (over the set $\mathcal{U}$) the set of all weak PR-trees equivalent to D has a corresponding set of graphs over the set $\mathcal{U}$ (i.e., the frontiers of the equivalent weak PR-trees).

Now, from the perspective of path-tree models, the relevant subset of equivalent weak PR-trees consists of the weak PR-trees with an acyclic frontier (i.e., the PR-trees). For a given weak PR-tree D we refer to this relevant subset as ***the equivalence class*** of a weak PR-tree and use $EQ(\text{D})$ to denote it (defined below). Furthermore, we refer to the set of trees that correspond to $EQ(\text{D})$ as the ***consistent set of D*** (denoted $\boldsymbol{Consistent(D)}$). More specifically, our algorithm in section 4 is given a constraint set $\mathbb{S}_G$ from a graph G, and produces a PR-tree D where T is a path-model of G (i.e., $T \in \mathcal{T}_G$) iff there is a PR-tree, $D^* \in EQ(\text{D})$, such that $T = Frontier(D^*)$ (i.e., $T \in Consistent(\text{D})$).

We now note some key observations regarding equivalence. In particular, equivalence transformations only affect the order imposed by P-nodes and R′-nodes on their children. Also, the helly property (see def. 3.2.10) of weak PR-trees is preserved through the application of equivalence transformations, since it has no requirements on the orders imposed by the nodes in a weak PR-tree. However, in order for a weak R-node $\mathcal{R}$ to be an R-node, we require a very specific relationship amongst the orders of the R′-nodes in $\mathcal{R}_{R'}$ (in particular, we require $\mathcal{R}$ to be PWF). Therefore, we have the following observation.

Observation 3.4.1 *If D and D* are equivalent weak PR-trees, then:*
(1) D is helly iff D is helly; and*
(2) PWF of the weak R-nodes in D and D can be inconsistent. For example, consider the R-node from fig. 6 (on page 13): by applying $\eta^{reverse}$ to Q'_1 the result is the weak R-node in fig. 3 (on page 10) and this weak R-node is **not** and R-node. Similarly, PC can be inconsistent.*

For a given weak PR-tree D, we say that the ***equivalence class of D*** is EQ(D) = {D* : D* ≡ D and $Frontier$(D) is acyclic}. Furthermore, we are really only interested in the frontiers of these weak PR-trees. With this in mind, we define ***consistent set*** of a weak PR-tree D, denoted $Consistent$(D), as the set of frontiers of weak PR-trees in EQ(D) (i.e., $Consistent$(D) = { $Frontier(D^*)$: $D^* \in EQ$(D)}). We similarly define the directed equivalence class and consistent set; i.e., $\overrightarrow{EQ}$(D) = {D^* : D^* is a weak PR-tree, $D^* \equiv$ D, and $\overrightarrow{Frontier}(D^*)$ is acyclic } and $\overrightarrow{Consistent}$(D) = {$\overrightarrow{Frontier}(D^*)$: $D^* \in \overrightarrow{EQ}$(D)}.

Using these equivalence transformations we provide the connection between the weak PR-tree and Booth and Lueker's PQ-tree [2] (see observation 3.4.2). A PQ-tree is an ordered rooted tree whose non-leaf nodes are identified as P-nodes and Q-nodes and two PQ-trees are equivalent when one can be transformed into the other by permuting the children of each P-node and reversing the order imposed by a some of the Q-nodes.

Observation 3.4.2 *A weak PR-tree D with precisely one source that is not a weak R-node, (i.e., D = {A_0}, {R_0, R_1, ..., $R_{\gamma-1}$}) is a PQ-tree.*
*Note: we call such a weak PR-tree a **rooted PR-tree**.*

Proof:
Notice that, each of D's weak R-nodes is trivial (i.e., $(R_i)_{R'}$ = {Q} for some R′-node Q in D and $(R_i)_T$ is precisely the path described by Q). Thus, each pair (R_i, Q) is the same as one of Booth and Lueker's Q-nodes. Furthermore, the only difference between our P-nodes and the P-nodes in PQ-trees is that our P-nodes are allowed to have more than one parent. Thus, by replacing each (R_i, Q) in D with the appropriate Q-node, the result is a PQ-tree T since each P-node in D now has exactly one parent. Similarly, the PQ-trees equivalent to T are precisely the weak PR-trees equivalent to D.
QED.

Recall that by lemma 3.3.4 and theorem 3.3.5, the frontier of a weak R-node is acyclic iff it is an R-node and the directed frontier of a weak R-node is acyclic iff it is a strong R-node. Using these results along with part (1) of observation 3.4.1, we have the following corollary.

Corollary 3.4.3 *If D is an HW-PR-tree, then:*
$\forall\ D^* \in EQ(D)$, D^* is a PR-tree; and $\forall\ D^* \in \overrightarrow{EQ}(D)$, D^* is a strong PR-tree.

We now have the definitions required to state the major theorems of this paper. In particular, in section 5 we prove that for every graph G, there is a PR-tree (strong PR-tree) whose consistent set (directed consistent set) is $\mathcal{T}_G$ ($\overrightarrow{\mathcal{T}}_G$). Section 4. In the next subsection we present the final properties that we use to prove this major theorem. In particular, we discuss the uniqueness of least common ancestors in PR-trees.

3.5 Least Common Ancestors in weak PR-trees

In this section, we discuss least common ancestors in weak PR-trees. For the purpose of this paper we are only concerned with the least common ancestors of leaf sets (with respect to a given weak PR-tree). We first note that, in a weak PR-tree, a leaf set L might have more than one node that qualifies as a least common ancestor in the traditional sense (i.e., a node N whose leaf set contains L, where none of N's descendants' leaf sets contain L; equivalently, $L \subseteq L(N)$ and N has at least two children whose leaf sets contain elements of L) For example, when an R′-node Q is a ***traditional least common ancestor (TLCA)*** of a leaf set L its weak R-node Q_R is also a TLCA of L (as in fig. 7 below). Furthermore, since a weak PR-tree can have multiple source nodes, there can be leaf sets that do not have a common ancestor (i.e., as in fig. 7 below), and, thus, have no least common ancestor.

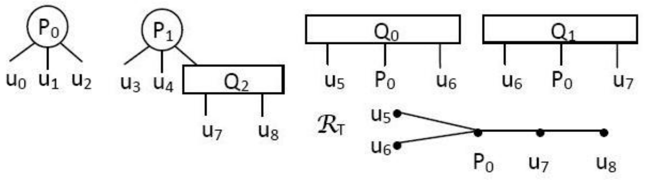

Fig. 7. A weak PR-tree where the leaf set $L_0 = \{u_0, u_3\}$ that has no common ancestor, and the leaf set $L_1 = \{u_0, u_6\}$ that has multiple TLCAs: $\mathcal{R}$, Q_0, and Q_1.

With these issues in mind, we consider HW-PR-trees and prove two interesting lemmas regarding TLCAs of leaf sets with common ancestors (see lemmas 3.5.1 and 3.5.2). These lemmas motivate our definition of a least common ancestor (see def. 3.5.3). In particular, for a leaf set L with respect to an HW-PR-tree, the LCA of L (when it exists) is unique and is either a P-node or a minimal subtree of the tree of an R-node. We then introduce a more general form of an LCA, which is defined for all leaf sets. In particular, we define the ***least common set of ancestors (LCSA)*** (see def. 3.5.5) of a leaf set L to be the set of least common ancestors of maximal subsets of L. Finally, we introduce the concept of a ***good leaf set*** with respect to a given PR-tree (see def. 3.5.7). For such a good leaf set L in a PR-tree D, we prove that the LCA of L exists and has L as its leaf set. The results from this section are extremely useful for the major theorem regarding PR-trees (i.e., that PR-trees characterize the sets of path-tree models of graphs).

Lemma 3.5.1 *For an HW-PR-tree D, a leaf set $L \subseteq L(D)$, and a P-node X in D: If X is a TLCA of L, then X is the unique TLCA of L in D.*

Proof:
Let X^* be a node in D that is a TLCA of L. Thus, $L \subseteq L(X) \cap L(X^*)$, and neither X nor X^* have a descendent whose leaf set contains L (i.e., $X \notin Desc(X^*)$ and $X^* \notin Desc(X)$); i.e., $\{X^*, X\}$ is a loop.

Furthermore, no descendent of X has a leaf set that contains $L(X) \cap L(X^*)$ (i.e., $\forall\ N \in Desc(X)$, $L(X) \cap L(X^*) \nsubseteq L(N)$). Therefore, this loop satisfies the helly property iff $X = X^*$ (since it must satisfy property (1) of def. 3.2.10). ***QED.***

We now prove a similar lemma regarding weak R-nodes.

Lemma 3.5.2 *For an HW-PR-tree D, $L \subseteq L(D)$, and a weak R-node $\mathcal{R}$ in D, if*

- *$\mathcal{R}$ is a TLCA of L; and*
- *$\mathcal{C}_L$ is the subset of $\mathcal{R}$'s children whose leaf sets have elements in common with L (i.e., $\forall N \in Children(\mathcal{R}), L(N) \cap L \neq \emptyset \Rightarrow N \in \mathcal{C}_L$); and*
- *$\mathcal{C}$ is the subset of $\mathcal{R}$'s children corresponding to the minimal subtree of $\mathcal{R}_T$ induced by $\mathcal{C}_L$ (i.e., $\mathcal{C}$ is minimal such that $\mathcal{C}_L \subseteq \mathcal{C}$ and $\mathcal{R}_T[\mathcal{C}]$ is a tree);*

then:

If X is a TLCA of L and X is not $\mathcal{R}$, then $X \in \mathcal{R}_{R'}$ and $\mathcal{C} \subseteq Children(X)$ (i.e., $\mathcal{R}_T[Children(X)]$ contains $\mathcal{R}_T[\mathcal{C}]$ as a subpath).

Note: as a corollary to this lemma we can see that the tree $\mathcal{R}_T[\mathcal{C}]$ is a unique substructure of every TLCA of L, and, consequently, can be considered the uniquely defined least common ancestor of L.

Proof:
Notice that, for any path: N_0, N_1, ..., N_{r-1}; in $\mathcal{R}_T$, there must be R′-nodes Q_0, Q_1, ..., $Q_{z-1} \in \mathcal{R}_{R'}$ such that $\{N_0, N_1, ..., N_{r-1}\} \subseteq \bigcup_{i=0}^{z-1} Children(Q_i)$. Furthermore, (w.l.o.g.) we can select a minimal set $\{Q_0, Q_1, ..., Q_{z-1}\}$ that is ordered so that: $N_0 \in Children(Q_0)$, $N_{r-1} \in Children(Q_{z-1})$, and for every $i \in [0, z-2]$, $Children(Q_i) \cap Children(Q_{i+1}) \neq \emptyset$.

Let $\mathcal{C}_L$ and $\mathcal{C}$ be defined as in the statement of this lemma, and let $\mathcal{C} = \{C_0, C_1, ..., C_{k-1}\}$. Consider (w.l.o.g.) the path C_0, C_1, ..., C_{r-1} (for $2 \leq r \leq k$) in $\mathcal{R}_T$ where $C_0, C_{r-1} \in \mathcal{C}_L$, and $\forall j \in [1, r-2], C_j \notin \mathcal{C}_L$ (i.e., $L(C_j) \cap L = \emptyset$). Now, we let $\{Q_0, Q_1, ..., Q_{z-1}\}$ be a minimal subset of $\mathcal{R}_{R'}$ such that: $C_0 \in Children(Q_0)$, $C_{r-1} \in Children(Q_{z-1})$, and for every $i \in [0, z-2]$,

$Children(Q_i) \cap Children(Q_{i+1}) \neq \emptyset$.

We now consider the leaf set $L^* = L \cap (L(C_0) \cup L(C_{r-1}))$. Notice that $\mathcal{R}$ is a TLCA of L^*. We now show that any TLCA of L^*, which is not $\mathcal{R}$, must be a node in $\mathcal{R}_{R'}$, and, consequently, any TLCA of L is either $\mathcal{R}$ or a node in $\mathcal{R}_{R'}$ (since $L^* \subseteq L$ and $\mathcal{R}$ is a TLCA of both L and L^*).

Let X be a TLCA of L^* which is not $\mathcal{R}$ (note: by the existence of this TLCA we will see that $z = 1$). Thus, $\{Q_0$, ..., Q_{z-1}, $X\}$ is a loop. Furthermore, (by the helly property) $L(Q_0) \cap L(Q_{z-1}) \neq \emptyset$, and, consequently, $\{Q_0$,$Q_{z-1}\}$ is a loop. Therefore, (by the helly property) $L(Q_0) \cap L(Q_{z-1}) \neq \emptyset$, and, consequently, $Children(Q_0) \cap Children(Q_{z-1}) \neq \emptyset$ (since $Q_0, Q_{z-1} \in \mathcal{R}_{R'}$). Therefore, since z was chosen to be minimal, $z \leq 2$.

We now consider to the loop $\{Q_0$,Q_{z-1},$X\}$, and let $L^{**} = L(Q_0) \cap L(Q_{z-1}) \cap L(X)$ (note: by the helly property $L^{**} \neq \emptyset$). Suppose (for a contradiction) that $z = 2$ (i.e., $Q_{z-1} = Q_1$). By the minimality of z, we know that C_0 is not a child of Q_1 and C_{r-1} is not a child of Q_0. Thus, (since $L(Q_0) \cap L(Q_1) \neq \emptyset$) we must have $i \in [1, r-2]$ such that $L(C_i) \cap L^{**} \neq \emptyset$ (i.e., $L(X) \cap L(C_i) \neq \emptyset$). We now consider the loop $\{Q_0$, $X\}$. Notice that, in order for this loop to satisfy the helly property we must have a node $N \in \mathcal{R}_{R'}$ where: $\{C_0, C_1, ..., C_i\} \subseteq Children(N)$, and either $N = X$ or $N \in Desc(X)$. Similarly, (via the loop $\{Q_1$,$X\}$) we must have a node $N^* \in \mathcal{R}_{R'}$ where: $\{C_i, C_{i+1}, ..., C_{r-1}\} \subseteq Children(N^*)$, and either $N^* = X$ or $N^* \in Desc(X)$. Therefore, we have a node $N^{**} \in \mathcal{R}_{R'}$ where: $\{C_0$, C_1, ..., $C_{r-1}\}$ $\subseteq Children(N^{**})$, and either $N^{**} = X$ or $N^{**} \in Desc(X)$ (i.e., N^{**} satisfies both of the previous statements), and, consequently, we contradict the minimality of z (since $\{N^{**}\}$ can be used in place of $\{Q_0$,$Q_1\}$). Thus, $z = 1$.

Since $z = 1$, C_0 and C_{r-1} must be children of Q_0; therefore, $L^* \subseteq L(X) \cap L(Q_0) = L^{**}$. Furthermore, neither X nor Q_0 have a descendent whose leaf set contains L^* (i.e., $Q_0 \notin Desc(X)$ and $X \notin Desc(Q_0)$). Thus, the only way the loop $\{Q_0$,$X\}$ can satisfy the helly property is if $X \in \mathcal{R}_{R'}$ and $\{C_0$, C_1, ..., $C_{r-1}\} \subseteq Children(X) \cap Children(Q_0)$, i.e., via property (2) of def. 3.2.10.
QED.

The two previous lemmas lead to the following definition of the least common ancestor of a leaf set in an HW-PR-tree.

Definition 3.5.3 *For an HW-PR-tree D and a subset L of the leaves of D, X is **the least common ancestor (LCA) of L in D**, when one of the following holds:*
(1) $|L| = 1$, and $X \in L$ (i.e., X is the only element in L); or
(2) X is a P-node, $L \subseteq L(X)$, and X has at least two children whose leaf sets contain elements of L; or
(3) X is a minimal subtree of the tree of a weak R-node, such that $L \subseteq L(X)$, and all of X's children have leaf sets that contain elements of L.

Note: when X is a subtree of the tree of a weak R-node $\mathcal{R}$ we define $Frontier(X)$

to be $Frontier(\mathcal{R})[L(X)]$. Also, we use $\boldsymbol{LCA(D, L)}$ to denote L's LCA in D.

This definition (along with lemmas 3.5.1 and 3.5.2) tells us that, when an LCA of a given leaf set exists, it will be unique (i.e., as in the following corollary). Notice that, while weak R-nodes and R′-nodes can be TLCAs of a leaf set, they cannot be LCAs of a leaf set. In particular, when a weak R-node $\mathcal{R}$ or R′-node Q is a TLCA of a leaf set L, the LCA of that leaf set will be a minimal subtree of the tree of $\mathcal{R}$ or Q_R (respectively). For example, in fig. 7 above, the leaf set L_1 has three TLCAs: Q_0, Q_1, and $\mathcal{R}$; and the LCA of L_1 is the path: P_0, u_6; in $\mathcal{R}_T$.

Corollary 3.5.4 *For an HW-PR-tree D and $L \subseteq L(D)$, if $LCA(D, L)$ exists, then it is unique.*

Since the LCA is undefined for certain leaf sets (i.e., as in fig. 7), we define of the ***least common set of ancestors (LCSA)*** of a given leaf set. In particular, the LCSA exists for every leaf subset of an HW-PR-tree.

Definition 3.5.5 *For an HW-PR-tree D and a subset L of the leaves of D, we say that $\{X_0, X_1, ..., X_{k-1}\}$ is **the least common set of ancestors (LCSA) of L in $\boldsymbol{D}$**, denoted $\boldsymbol{LCSA(D, L)}$, when:*

(1) For every $L^* \subseteq L$, if L^* is maximal such that $LCA(D, L^*)$ exists, then $LCA(D, L^*) \in LCSA(D, L)$; and

(2) For every $i \in [0, k-1]$:
- *$L(X_i) \cap L$ is maximal such that $LCA(D, L(X_i) \cap L)$ exists; and*
- *$X_i = LCA(D, L(X_i) \cap L)$.*

Notice that, when a leaf set L has an LCA X the LCSA of L will be $\{X\}$. Furthermore, by the uniqueness of LCAs in HW-PR-trees, LCSAs are also unique in HW-PR-trees. For example, in fig. 7 above, the leaf set $L_0 = \{u_0, u_3\}$, has $LCSA(D, L_0) = \{P_0, P_1\}$ (note: L_0 has no LCA in D).

Corollary 3.5.6 *For an HW-PR-tree D and $L \subseteq L(D)$, $LCSA(D, L)$ is unique.*

We will now introduce the concept of a "good" leaf set (see def. 3.5.7) such that, when a leaf set L from a PR-tree is good, the LCA of L exists and its leaf set is precisely L (see theorem 3.5.8 below).

Definition 3.5.7 *For a PR-tree D we say that L is a **good leaf subset of D** when: $L \subseteq L(D)$, and the frontier of every PR-tree $D^* \equiv D$ contains a path whose vertex set is L (i.e., $\forall T \in Consistent(D), T[L]$ is a path).*

Theorem 3.5.8 *For a PR-tree D, if L is a good leaf set in D, then $X = LCA(D, L)$ exists (i.e., $LCSA(D, L) = \{X\}$), and $L(X) = L$ (i.e., as in $(*)$ below).*

$$(*) \begin{cases} \text{(1) There is a leaf } X \text{ in } D\text{, such that: } X = LCA(D, L) \text{ (i.e., } |L| = 1\text{); or} \\ \text{(2) There is a P-node } X \text{ in } D \text{ such that: } X = LCA(D, L) \text{ and } L(X) = L\text{; or} \\ \text{(3) There is a path } X = C_0, C_1, ..., C_{k-1} \text{ in the tree of an R-node } \mathcal{R} \text{ such} \\ \quad\quad \text{that: } X = LCA(D, L) \text{ and } L(X) = L. \end{cases}$$

Proof:

Let L = $\{u_0, u_1, ..., u_{n-1}\}$ be a good leaf set in D. Thus, $Frontier(D)[L]$ is a path and (w.l.o.g.) we let P = $u_0, u_1, ..., u_{n-1}$ be this path. We prove that L satisfies $(*)$ by induction on the size of L.

Notice that, when n = 1, the leaf X in L is the LCA of L (i.e., $L(X) = \{X\} = L$). Thus, satisfying property (1) of $(*)$. Now consider $|L| > 1$ and assume:

(IH) For every $L^* \subseteq L(D)$, if L^* is a good leaf set with respect to D and $|L^*| < |L|$, then L^* satisfies $(*)$.

We prove the inductive case in two stages. In the first stage we prove that L has an LCA (i.e., that $|LCSA(D, L)| = 1$). We complete the proof by showing that the leaf set of every child of $LCA(D, L)$ is contained in L (i.e., $L(LCA(D, L)) \subseteq L$).

[Stage 1:] We now assume (for a contradiction) that $LCSA(D, L) = \{X_0, X_1, ..., X_{r-1}\}$, where $r > 1$ (i.e., $LCA(D, L)$ is not defined). Also, for every $i \in [0, r-1]$, we let $L_i = L(X_i) \cap L$.

Notice that, for every $i \in [0, r-1]$, $Frontier(X_i)[L_i]$ must be a subpath of $Frontier(D)[L]$. In particular, since $Frontier(D)[L]$ is a path, if $Frontier(X_i)[L_i]$ was not a subpath of $Frontier(D)[L]$, then we would have a cycle formed by $Frontier(D)[L] \cup Frontier(X_i)$, and, consequently, we would contradict the fact that the frontier of a PR-tree is a tree (i.e., as in theorem 3.3.3). Therefore, for every $i \in [0, r-1]$, $Frontier(X_i)[L_i]$ must be a subpath of $Frontier(D)[L]$. Furthermore, (since L is a good leaf set) in every PR-tree $D^* \equiv D$, the LCA X_i^* corresponding to X_i, must also satisfy this condition (i.e., $Frontier(X_i^*)[L_i]$ must be a subpath of $Frontier(D^*)[L]$). Therefore, for every $i \in [0, r-1]$, L_i is a good leaf set with respect to D, and, by (IH), $L(X_i) = L_i$. In particular, we must have L_i = $\{u_{j_i}, u_{j_i+1}, ..., u_{k_i}\}$, where $0 \leq j_i \leq k_i \leq n-1$. Also, by the maximality of L_i, we know that, for any $i^* \in [0, r-1], i^* \neq i$, L_i is not contained in L_{i^*}.

Consider an edge $e = u_\ell u_{\ell+1}$ (for $\ell \in [0, n-2]$) on the path P. Notice that, we must have a node N where e is an edge in the frontier of N (i.e., N is a common ancestor of u_ℓ and $u_{\ell+1}$). Therefore, for every edge $e = u_\ell u_{\ell+1}$ (for $\ell \in [0, n-2]$), we must have some $i \in [0, r-1]$ such that the frontier of X_i contains the edge e (i.e., $j_i < k_i$).

Therefore, (w.l.o.g.) we order $\{X_0, X_1, ..., X_{r-1}\}$ such that, for every $i \in [0, r-1]$, $Frontier(X_i)$ is the path: $u_{j_i}, u_{j_i+1}, ..., u_{k_i}$, where:

$j_0 = 0$, $k_{r-1} = n$, and $j_i < j_{i+1} \leq k_i < k_{i+1}$.

Furthermore, we let $Children(X_0)$ = $\{N_0, N_1, ..., N_{z-1}\}$, such that (w.l.o.g) $N_0 < N_1 < ... < N_{z-1}$. We now consider the two possible cases for X_0 (i.e., either X_0 is a P-node or X_0 is a minimal subpath [8] of an R-node), and demon-

[8] Otherwise: $Frontier(X_0)$ (i.e., $Frontier(\mathcal{R})[L_0]$) would not be a path (i.e., contradicting the fact that L_0 is a good leaf set).

strate an equivalence transformation ψ on D such that: $\psi(D)$ is a PR-tree, and $Frontier(\psi(D))[L]$ is not a path (i.e., contradicting the fact that L is a good leaf set). Note: by observation 3.4.1, any equivalence transformation we apply to D will result in an HW-PR-tree (since D is helly); therefore, to ensure that $\psi(D)$ is a PR-tree we just need to make sure that every weak R-node in $\psi(D)$ is an R-node.

[Stage 1: Case 1] X_0 is a P-node.

Since X_0 is a P-node, $z > 2$ (i.e., X_0 has at least three children). Furthermore, by the helly property and the order of $LCSA(D, L)$, $L(X_0) \cap L(X_1) \subseteq L(N_{z-1})$. We now consider the equivalence transformation ψ on D where ψ changes the order X_0 imposes on its children to be $N_0 < N_{z-1} < N_1 <$ ... N_{z-2} (and nothing else). Notice that, since ψ only affects a single P-node from the PR-tree D, all of the R-nodes in D are still R-nodes in $\psi(D)$. Thus, $\psi(D)$ is a PR-tree. However, since X_1 and its descendents are unchanged by ψ, $Frontier(\psi(X_1)) = Frontier(X_1)$. Moreover, u_{k_0} is adjacent to u_{k_0+1} in $Frontier(\psi(D))[L]$ (note: u_{k_0+1} is not a leaf of X_0). Additionally, u_{k_0} is an internal vertex with respect to the frontier of $\psi(X_0)$ (i.e., in $Frontier(\psi(D))[L]$, u_{k_0} is adjacent to two leaves: u_ℓ, u_{ℓ^*}; of X_0). Therefore, $Frontier(\psi(D))[L]$ contains the claw: $\{u_{k_0}; u_{k_0+1}, u_\ell, u_{\ell^*}\}$ (where u_{k_0+1} is the centre vertex, and the others are leaves), and, consequently, $Frontier(\psi(D))[L]$ is not a path (contradicting the fact that L is a good leaf set).

[Stage 1: Case 2] X_0 is a subpath of the tree of an R-node $\mathcal{R}$.

Note, from the order of $LCSA(D, L)$, we must have $i \in [\boldsymbol{1}, z-1]$, such that the leaves: u_{j_1}, ..., u_{k_0}; are contained in the leaf sets of the last $z-i$ children of X_0 (i.e., $L(X_0) \cap L(X_1) \subseteq \bigcup_{j=i}^{z-1} L(N_j)$). We now consider subcases regarding the value of i and $|L(N_{z-1})|$.

Case 1.2.(a) $i < z - 1$.

We first show that we must have an R′-node $Q^* \in Desc(X_1)$ where $Children(Q)$ = $\{N_i, ..., N_{z-1}\}$. We will then demonstrate an equivalence transformation ψ on D such that $\psi(D)$ is a PR-tree and $Frontier(\psi(D))[L]$ contains a claw.

By the helly property, either:

- X_1 is a path in an R-node such that $\{N_i, N_{i+1}, ..., N_{z-1}\} \subseteq Children(X_1)$ (i.e., X_1 is a path in $\mathcal{R}$); or
- There is an R′-node $Q^* \in Desc(X_1)$, such that $\{N_i, N_{i+1}, ..., N_{z-1}\} \subseteq Children(Q^*)$ (i.e., $Q^*_R = \mathcal{R}$).

Notice that, since L_1 must be maximal, if there is no such Q^*, X_1 must have a child N_z distinct from the children of X_0. Furthermore, by the order of $LCSA(D, L)$, (w.l.o.g.) N_z is adjacent to N_{z-1} in $\mathcal{R}_T$. However, this would contradict the maximality of X_0 since the path $N_0, N_1, ..., N_{z-1}, N_z$ contains X_0 as a subpath and has a greater intersection with L. Also, when such a Q^* exists, the children of Q^* must be contained in $\{N_i, N_{i+1}, ..., N_{z-1}\}$; otherwise, we would similarly contradict the maximality of L_0. Thus, Q^*

$\in Desc(X_1)$ where $Children(Q^*) = \{N_i, N_{i+1}, ..., N_{z-1}\}$.

Consider the equivalence transformation ψ on D, where: for every R′-node Q in $\mathcal{R}_{R'}$, ψ reverses the order Q imposes on its children (and nothing else). Notice that, since $\mathcal{R}$ is PWF and all of the R′-nodes in $\mathcal{R}_{R'}$ have the order of their children reversed by ψ, $\psi(\mathcal{R})$ is also PWF (i.e., $\psi(\mathcal{R})$ an R-node). Furthermore, since $\mathcal{R}$ is the only R-node that is affected by ψ and every weak R-node in D is PWF, every weak R-node in $\psi(D)$ will be PWF (i.e., an R-node). Thus, $\psi(D)$ is a PR-tree. Also, since ψ only affects nodes in $\mathcal{R}_{R'}$, for all $j \in [0, z-1]$, $\psi(N_j) = N_j$.

We will now demonstrate the existence of a claw, $\{u_\ell; u_{k_0+1}, u_{\ell^*}, u_{\ell^{**}}\}$, in $Frontier(\psi(D))[L]$ to contradict the fact that L is a good leaf set.

Suppose (w.l.o.g.) that Q orders its children: $N_i < N_{i+1} < ... < N_{z-1}$; then, $\psi(Q)$ will have: $N_{z-1} < N_{z-2} < ... < N_i$. More specifically, $\sigma(\psi(Q))$ = $\sigma(N_{z-1})$, $\sigma(N_{z-2})$, ..., $\sigma(N_i)$. Let u_ℓ be the last element of $\sigma(N_i)$ (i.e., u_ℓ is also the last element of $\sigma(\psi(Q))$). We consider two subcases:

(1) When $|L(N_i)| > 1$: we let $u_{\ell^*} = u_{\ell-1}$ (i.e., $u_{\ell^*} \in L(N_i)$); then:

- $\sigma(Q) = \overbrace{u_{j_1}, ..., u_{\ell^*}, u_\ell}^{\sigma(N_i)}, ..., \overbrace{..., u_{k_0}}^{\sigma(N_{z-1})}$; and
- $\sigma(\psi(Q)) = \underbrace{..., u_{k_0}}_{\sigma(N_{z-1})}, ..., \underbrace{u_{j_1}, ..., u_{\ell^*}, u_\ell}_{\sigma(N_i)}$.

(2) When $|L(N_i)| = 1$ (note: $u_\ell = u_{j_1}$): we let u_{ℓ^*} be the last element of $\sigma(N_{i+1})$; then:

- $\sigma(Q) = \overbrace{u_\ell}^{\sigma(N_i)}, \overbrace{..., u_{\ell^*}}^{\sigma(N_{i+1})}, ..., \overbrace{..., u_{k_0}}^{\sigma(N_{z-1})}$; and
- $\sigma(\psi(Q)) = \underbrace{..., u_{k_0}}_{\sigma(N_{z-1})}, ..., \underbrace{..., u_{\ell^*}}_{\sigma(N_{i+1})}, \underbrace{u_\ell}_{\sigma(N_i)}$.

Note, in both cases: u_{ℓ^*} is adjacent to u_ℓ, and u_ℓ, $u_{\ell^*} \in L(X_0) \cap L(X_1)$. Furthermore, $\sigma(X_1) = \sigma(Q), u_{k_0+1}, ..., u_{k_1}$; and $\sigma(\psi(X_1)) = \sigma(\psi(Q)), u_{k_0+1}, ..., u_{k_1}$ (i.e., the last vertex u_ℓ of $\sigma(\psi(Q))$ is adjacent to u_{k_0+1}). Also, $u_{k_0+1} \neq u_\ell$ and $u_{k_0+1} \neq u_{\ell^*}$ (since $u_{k_0+1} \notin L(X_0)$).

Now, let $u_{\ell^{**}}$ be the first element of $\sigma(N_{i-1})$; then (w.l.o.g.):

- $\sigma(X_0) = u_0, ..., \overbrace{u_{\ell^{**}}, ...}^{\sigma(N_{i-1})}, \sigma(N_i), ..., \sigma(N_{z-1})$; and
- $\sigma(\psi(X_0)) = \sigma(N_{z-1}), ..., \sigma(N_i), \underbrace{u_{\ell^{**}}, ...}_{\sigma(N_{i-1})}, ..., u_0$.

Note: in $Frontier(\psi(X_0))$, $u_{\ell^{**}}$ is adjacent to u_ℓ (since u_ℓ is the last element of $\sigma(\psi(N_i))$). Also, $u_{\ell^{**}} \notin \{u_{k_0+1}, u_\ell, u_{\ell^*}\}$ (since $u_{\ell^{**}} \notin L(X_1)$).

Therefore u_{k_0+1}, u_ℓ, u_{ℓ^*}, and $u_{\ell^{**}}$ are distinct leaves such that: u_ℓ is adjacent to u_{k_0+1}, u_{ℓ^*}, and $u_{\ell^{**}}$ in $Frontier(X_0) \cup Frontier(X_1)$, and, consequently, $Frontier(\psi(D))[L]$ contains a claw.

Case 1.2.(b) $i = z-1$ (i.e., $L(X_0) \cap L(X_1) \subseteq L(N_{z-1})$) and $|L(N_{z-1})| > 1$.

Notice that, X_1 cannot be a subpath of $\mathcal{R}$ (otherwise, we could extend the path X_0 using the path X_1[9], and, thus, contradict the maximality of

[9] Since, by the definition of weak R-nodes, X_1 would need to have N_{z-1} as a child.

L_0). Similarly, X_1 cannot have a descendent in $\mathcal{R}_{R'}$ (otherwise, we could similarly contradict the maximality of L_0). Now, consider applying ψ as in case (a) to D (i.e., ψ reverses the order imposed by each R′-node in $\mathcal{R}_{R'}$, and nothing else). Note: $\psi(D)$ is a PR-tree as in case (a).

We now show that, in $Frontier(\psi(D))[L]$, we will have the claw $\{u_{k_0}; u_{k_0-1}, u_{k_0+1}, u_\ell\}$; where, u_ℓ is the first element of $\sigma(N_{z-2})$. In particular, since X_1 is not a path in $\mathcal{R}_T$ and does not have any descendents in $\mathcal{R}_{R'}$, $Frontier(\psi(X_1)) = Frontier(X_1)$. Therefore, in $Frontier(\psi(X_1))$, u_{k_0} is still adjacent to u_{k_0+1} (note: u_{k_0+1} is the first non-X_0 leaf of X_1). Furthermore, since $|L(N_{z-1})| > 1$ and N_{z-1} is not affected by ψ, u_{k_0-1} and u_{k_0} are still adjacent in $Frontier(\psi(X_0))$. Finally, since we have reversed the order of every R′-node in $\mathcal{R}_{R'}$ and $|L(N_{z-1})| > 1$, we now have $N_{z-1} < N_{z-2}$, and, consequently, in $Frontier(\psi(X_0))$, we have u_{k_0} adjacent to the first element, u_ℓ, of $\sigma(N_{z-2})$. Therefore, in $Frontier(\psi(D))[L]$, we have the claw: $\{u_{k_0}; u_{k_0-1}, u_{k_0+1}, u_\ell\}$.

Case 1.2.(c) $i = z-1$ and $|L(N_{z-1})| = 1$ (i.e., $N_{z-1} = u_{k_0} = u_{j_1}$).

Notice that, since $N_{z-1} = u_{k_0}$, N_{z-1} will be a descendent of X_1. Also, $X_1 \neq N_{z-1}$; otherwise, we would contradict the maximality of L_1.

Consider the parent Y of u_{k_0} with respect to X_1.

Suppose that Y is an R′-node. Now, Y has a child in common with $\mathcal{R}$, and, consequently, $Y_R = \mathcal{R}$. Furthermore, Y must have at least two children: u_{k_0} and a node N (where, N is adjacent to u_{k_0} in $\mathcal{R}_T$). In particular, by adding the node N to the children of X_0, we would be able to extend the path X_0 so that L_0 includes additional elements of L. Therefore, we have contradicted the maximality of L_0. Similarly, if Y is a subpath of the tree of an R-node (i.e., $Y = X_1$), the R-node would have to be $\mathcal{R}$, and Y would have at least two children. Thus, we can similarly extend X_0, and, as such, Y would contradict the maximality of L_0.

Therefore, Y must be a P-node. Now, as in Case 1 (above), we apply an equivalence transformation ψ to re-order the children of Y so that u_{k_0} is an internal child with respect to the order imposed by $\psi(Y)$ (i.e., we have N,$N^* \in Children(Y)$ such that N and N^* are adjacent to u_{k_0} with respect to the order imposed by $\psi(Y)$). Now, in $Frontier(\psi(D))[L]$, we have:

- u_{k_0} adjacent to two leaves (i.e., u_ℓ and u_{ℓ^*} of N and N^* respectively) in $Frontier(\psi(Y))$ (since u_{k_0} is an internal child of Y); and
- u_{k_0} adjacent to u_{k_0-1} (since $\psi(X_0) = X_0$); and
- $u_{k_0-1} = u_{j_1-1}$ (i.e., u_{k_0-1} is not a leaf of Y).

Thus, in $Frontier(\psi(D))[L]$, we have the claw $\{u_{k_0}; u_{k_0-1}, u_\ell, \text{and } u_{\ell^*}\}$.

Thus, if X_0 is a subtree of the tree of an R-node, we have a contradiction.

Therefore, $|LCSA(D, L)| = 1$. ***[End of Stage 1]***

In particular, L must have an LCA X in D, i.e.:

- $L \subseteq L(X)$; and
- X is either a P-node or a path in the tree of an R-node; and

- There is a subset $\mathcal{C} = \{C_0, C_1, ..., C_{k-1}\}$ of the children of X such that:
  · $\forall i \in [0, k-1]$, $L(C_i) \cap L \neq \emptyset$; and
  · $L \subseteq \bigcup_{i=0}^{k-1} L(C_i)$; and
  · $k > 1$ (if $k = 1$, then X would not be an LCA).

Note: since $|\mathcal{C}| > 1$, X has at least two children C_0, C_1 with leaves in L.

Observe that, $Frontier(X)[L]$ must be the path $Frontier(D)[L]$. In particular, since $Frontier(D)[L]$ is a path, when $Frontier(X)[L] \neq Frontier(D)[L]$, there is a cycle in $Frontier(D)$ formed by $Frontier(D)[L] \cup Frontier(X)$ and, consequently, we have contradicted the fact that the frontier of a PR-tree is a tree (see theorem 3.3.3). Therefore $Frontier(X)[L]$ must be a path. Furthermore:

$(**)$ For every equivalence transformation ψ on D where $\psi(D)$ is a PR-tree: $Frontier(\psi(X))[L]$ must be a path.

[Stage 2:] We now show that the leaf set of every child of X must be contained in L and, consequently, $L(X) = L$.

[Stage 2: part 1] Suppose (for a contradiction) that N is a child of X whose leaf set is disjoint from L (i.e., $L(N) \cap L = \emptyset$). We consider two cases:

Case 2.1.(a) X is a P-node.
Consider an equivalence transformation ψ on D where the children of X are re-ordered so that: ... $< C_0 < N < C_1 <$ ... (i.e., we have permuted the children of the P-node X so that N falls between C_0 and C_1); and nothing else is changed. Thus, since ψ only modifies a single P-node in the PR-tree D, $\psi(D)$ is a PR-tree (i.e., as in Case 1 above). Furthermore, $Frontier(\psi(X))[L]$ is not connected. However, this contradicts $(**)$ above. Therefore, X cannot have a child whose leaf set has no elements in common with L.

Case 2.1.(b) X is a path in the tree of an R-node $\mathcal{R}$.
Notice that, if N were to be an end-point of X, then we could have chosen a shorter X as the LCA (providing a contradiction). Thus, N must be an internal node of X. Furthermore, $Frontier(\mathcal{R})[L]$ is not connected, and consequently, $Frontier(X)[L]$ is not connected. However, this contradicts $(**)$ above. Therefore, X cannot have a child whose leaf set has no elements in common with L.

Therefore every child of X must have leaves in L.

[Stage 2: part 2] Suppose (for a contradiction) that N is a child of X and some of N's leaves are not in L (note: N is in $\mathcal{C}$ since $L(N) \cap L \neq \emptyset$). W.l.o.g. let N be C_0 and let C_1 be adjacent to C_0 such that $C_0 < C_1$ with respect to the order imposed by X. We now consider two cases regarding C_0 (note: C_0 is not a leaf since it has at least one leaf in L and one leaf not in L):

Case 2.2.(a) C_0 is a P-node.

Consider an equivalence transformation ψ on D where the order of the children of C_0 is reversed and nothing else is changed. Thus, since ψ only modifies a single P-node in the PR-tree D, $\psi(D)$ is a PR-tree (i.e., as in Case 1 above). Furthermore, by reversing the children of C_0, there is a leaf $u_0 \in L(C_0) - L$ that is between a leaf $u_1 \in L \cap L(C_0)$ and a leaf $u_2 \in L \cap L(C_1)$ with respect to $Frontier(\psi(X))$. Thus, $Frontier(\psi(X))[L]$ is not connected and, consequently, we have contradicted $(**)$. Therefore, the leaf set of every P-node child of X must be contained in L.

Case 2.2.(b) C_0 is an R′-node and Let $\mathcal{R}$ = $(C_0)_R$.

Consider the equivalence transformation ψ on D where the order imposed by each R′-node in $\mathcal{R}_{R'}$ is reversed (note: the order imposed by C_0 is reversed). Now, since $\mathcal{R}$ is an R-node and ψ reverses the order of every R′-node in $\mathcal{R}_{R'}$, $\psi(\mathcal{R})$ is PWF (i.e., as in Case 2(a) above). Therefore, $\psi(D)$ is a PR-tree since it is helly (by observation 3.4.1) and we have ensured that the only R-node affected (i.e., $\mathcal{R}$) will remain PWF. Furthermore, with respect to $Frontier(X)$, ψ will only affect $\sigma(C_0)$ (since, by the helly property, only one descendent of X can belong to $\mathcal{R}_{R'}$). Thus, as in Case 2.2.(a), we can see that $Frontier(\psi(X))[L]$ is not a path. Therefore, we have contradicted $(**)$, and, consequently, the leaf set of every R′-node child of X must be contained in L.

Therefore the leaf set of every child of X is contained in L (i.e., $L(X) \subseteq L$).
[End of Stage 2].
QED.

We are now ready to demonstrate that the PR-tree characterizes the sets of path-tree models of graph. In particular, in section 4 we prove that: for every graph G, there is a PR-tree D such that $Consistent(D)$ = $\mathcal{T}_G$.

4 The PR-tree Construction Algorithm

In this section we present an algorithm to construct a PR-tree. In particular, given a graph G, first extract its constraint set $\mathbb{S}_G$, then use $\mathbb{S}_G$ to construct a PR-tree D with $Consistent(D) = \mathcal{T}_G$. We present the strong PR-tree construction algorithm alongside this algorithm since they are nearly identical (we will identify the differences, but focus on PR-tree construction).

We begin by discussing a restricted form of this problem (where G is connected and chordal). In the next subsection we provide an overview of our algorithm. This is followed by a detailed discussion of our algorithm's key steps. The correctness and time complexity are presented in sections 5 and 6 respectively. As mentioned in section 3, the HW-PR-tree plays an integral role in our algorithm. With this in mind, we use the following terminology to describe when an HW-PR-tree ***represents*** a path-tree problem, and when a node in an HW-PR-tree ***satisfies a constraint*** of a path-tree problem (see def. 4 below). Also, we use the following notation throughout this section.

For a graph G, let $\mathbb{S} = \mathbb{S}_G = \{S_0, S_1, ..., S_{n-1}\}$, for $i \in [1, n]$:

- Let $\mathcal{U}_i$ be the elements of S_0, ..., S_{i-1} (i.e., $\mathcal{U}_i = \bigcup_{j=0}^{i-1} S_j$), and
- Let $\mathbb{S}_i$ be the first i constraints of $\mathbb{S}$ (i.e., $\mathbb{S}_i = \{S_0, S_1, ..., S_{i-1}\}$), and
- Let S_i^* be the subset of S_i contained in $\mathcal{U}_i$ (i.e., $S_i^* = S_i \cap \mathcal{U}_i$).
- Let $\mathcal{T}_{\mathbb{S}_i}$ be the set of trees over $\mathcal{U}_i$ satisfying the first i constraints. (i.e., $\mathcal{T}_{\mathbb{S}_i} = \{T : T$ is a tree, $\mathrm{V}(T) = \mathcal{U}_i$, and $T[S_j]$ is a path for all $j \in [0, i-1]\}$).

Definition 4 *For a constraint set $\mathbb{S}$ and HW-PR-tree D, we say that:*

(1) A P-node or R′-node ***N*** *in D* ***satisfies a constraint S*** *when $S \cap L(N)$ is a good leaf set with respect to N (i.e., for every equivalence transformation ψ on D, $Frontier(\psi(N))[S \cap L(N)]$ is a path).*

(2) ***D represents*** *$\mathbb{S}$ when $Consistent(D) = \mathcal{T}_{\mathbb{S}}$ and every P-node or R′-node N in D satisfies every constraint in $\mathbb{S}$.*

4.1 Connected Chordal path-tree problems

We now demonstrate that we can restrict our focus to connected chordal graphs. Furthermore, we provide a variation (algorithm 2) on the general `Reduction` algorithm (algorithm 1 in section 1) for this restricted case.

Consider a graph G with connected components H_0, H_1, ..., H_{k-1}. Notice that, $T \in \mathcal{T}_G$ iff there are T_0, T_1, ..., T_{k-1} where $T_i \in \mathcal{T}_{H_i}$ such that T is obtained by adding edges to the forest T_0, T_1, ..., T_{k-1}. In particular, when G is not connected, $\mathcal{T}_G$ is fully specified by its connected components (i.e., we only need to consider connected graphs). Furthermore, since the path graphs (i.e., the graphs G where $\mathcal{T}_G \neq \emptyset$) are strictly contained within the chordal graphs, if G is not chordal then $\mathcal{T}_G = \emptyset$.

Also, testing for chordality can be performed in linear time (i.e., $\mathcal{O}(n + m)$) [20]. Additionally, via lexicographic breadth first search (LexBFS), the maximal clique vs. vertex incidence matrix of a chordal graph (i.e., $\mathbb{S}_G$) can be determined in linear time. Using these two facts, we have a pre-processing step (denoted `PreProc(G)`) that takes a connected graph G as input, produces $\mathbb{S}_G$ as output, and executes in $\mathcal{O}(n + m)$ time. In particular, we check the chordality of G (returning $\emptyset$ when G is not chordal) and then extract $\mathbb{S}_G$ via LexBFS. Furthermore, since G is connected, we order the constraint set $\mathbb{S}_G = \{S_0, S_1, ..., S_{n-1}\}$ such that for all $i \in [1, n]$ $G[\{v_0, v_1, ..., v_{i-1}\}]$ is connected (i.e., $\forall i \in [1, n-1], \exists j \in [0, i-1] : S_i \cap S_j \neq \emptyset$). Notice that, a breadth-first or depth-first search on the G provides such an ordering.

Using the output $\mathbb{S} = \{S_0, S_1, ..., S_{n-1}\}$ of `PreProc(G)` as input, we provide the following variation (see algorithm 2) of the general `Reduction` approach given in the introduction (see algorithm 1 in section 1). In particular, this variation relies on the "connected order" of $\mathbb{S}_G$ (i.e., for every $i \in$

$[1, n]$ there exists some $j \in [0, i-1]$, where $S_i \cap S_j \neq \emptyset$). This is the basis for our algorithm and uses the following relationship between $\mathcal{T}_{\mathbb{S}_i}$ and $\mathcal{T}_{\mathbb{S}_{i+1}}$.

Observation 4.1.1 *For a connected graph G, let $\mathbb{S}_G = \{S_0, S_1, ..., S_{n-1}\}$, where: $\forall i \in [1, n-1], \exists j \in [0, i-1] : S_i \cap S_j \neq \emptyset$. Then, if $\mathbb{S}_i^* = \mathbb{S}_i \cup \{S_i^*\}$, and $T \in \mathcal{T}_{\mathbb{S}_i}$:*

*(1) if $T[S_i^*]$ is not a path, then there is **no** $T^* \in \mathcal{T}_{\mathbb{S}_{i+1}}$ such that $T^*[\mathcal{U}_i] = T$.*

(2) Otherwise (i.e., $T[S_i^*]$ is a path), if T' is a tree with $V(T') = \mathcal{U}_{i+1}$ such that T is a subtree of T' (i.e., $T'[\mathcal{U}_i] = T$) and $T'[S_i]$ is a path, then $T' \in \mathcal{T}_{i+1}$.

Note: This justifies the loop invariant of algorithm 2 below.

Proof:
(1): Suppose (for a contradiction) that $T^* \in \mathcal{T}_{\mathbb{S}_{i+1}}$ such that $T^*[\mathcal{U}_i] = T$. Note, $T[S_i^*]$ cannot contain a connected component which is not a path since T^* would then fail to satisfy S_i. Thus, $T[S_i^*]$ is a collection of disconnected paths. Now, for any pair of connected components of $T[S^*]$, there must be a path in T connecting them, and $T^*[S_i]$ must be a path. Thus, T^* contains a cycle (contradicting the fact that T^* is a tree). Therefore, proving (1).

(2): Notice that, T' satisfies the constraints $S_0, S_1, ..., S_{i-1}$, since $T'[\mathcal{U}_i] = T$. Also, T' satisfies S_i since $T'[S_i]$ is a path. Therefore, $T' \in \mathcal{T}_{\mathbb{S}_{i+1}}$.
QED.

Note: From here on we assume that every constraint set $\mathbb{S} = \{S_0, S_1, ..., S_{n-1}\}$ is chordal, connected, and ordered such that $\forall i \in [1, n-1], \exists j \in [0, i-1] : S_i \cap S_j \neq \emptyset$ (i.e., $\mathbb{S}$ is the output of `PreProc`(G) for a connected graph G).

Algorithm 2 ***`Reduction`*****(**$\mathbb{S}$**)**, ***version 2:***
Solving a connected chordal path-tree problem $\mathbb{S}$ via reduction.

```
#Pre: $\mathbb{S} = \mathbb{S}_G = \{S_0, S_1, ..., S_{n-1}\}$ for a connected chordal graph $G$.
#Post: $\mathcal{T}_n = \mathcal{T}_G$.
  Initialize $\mathcal{U}_0 = \emptyset$, $\mathcal{T}_0 = \emptyset$.
  for $i = 1..n$:  # invariant: $\mathcal{T}_i = \mathcal{T}_{\mathbb{S}_i}$
    $\mathcal{U}_i = \mathcal{U}_{i-1} \cup S_{i-1}$.
    $S_i^* = \mathcal{U}_i \cap S_i$.
    $\mathcal{T}_i^* = \mathcal{T}_{i-1} \setminus \{T \in \mathcal{T}_{i-1} : T[S_i^*]$ is not a path$\}$.
    $\mathcal{T}_i = \{T : T$ is a tree, $V(T) = \mathcal{U}_i, T[S_i]$ is a path, and $T[\mathcal{U}_{i-1}] \in \mathcal{T}_i^*\}$.
  return $\mathcal{T}_n$
```

4.2 Overview of the algorithm

In this section we re-write algorithm 2 using PR-trees in place of the collections of trees (see algorithm 3 below). This will require operations corresponding to lines 7 and 8 that will produce an appropriate PR-tree. Rather than using PR-trees directly, we instead use HW-PR-trees. We do this so that we do not need to concern ourselves with maintaining pairwise wellformedness as we iteratively introduce constraints. In particular, the algorithm first constructs an HW-PR-tree D representing $\mathbb{S}$, then transforms D into a PR-tree (i.e., by applying equivalence transformations to the weak R-nodes in D). We refer to this final step as the operation ***fix_Rnodes(D)*** and it will return a PR-tree equivalent to D when one exists, and an empty PR-tree otherwise (the details of this operation are presented in section 4.4).

Notice that, for line 7, we need an operation that takes: an HW-PR-tree D that represents a constraint set $\mathbb{S}$ and a subset S^* of D's leaves (i.e., $S^* \subseteq L(D)$) as input, and produces an HW-PR-tree D^* that represents $\mathbb{S}^* = \mathbb{S} \cup \{S^*\}$. We use ***reduce(D, S*)*** to denote this operation. This method is quite complex and is discussed in detail in section 4.3. Notice that, S^* will be a good leaf set in D^*. Therefore, as long as there is a PR-tree D^{**} which is equivalent to D^* (i.e., $Consistent(D^*) \neq \emptyset$), by theorem 3.5.8, S^* must have a unique LCA in D^*. This LCA will be useful for our operation corresponding to line 8. Furthermore, when the LCA of S^* is a path in the tree of an R-node $\mathcal{R}$, `reduce` will create an R′-node $Q \in \mathcal{R}_{R'}$ which is precisely this path. Thus, in D^*, the unique LCA X of S^* will either be a P-node or an R′-node. In particular, `reduce`(D,S^*) will return $(0, 0)$ when $\mathcal{T}_{\mathbb{S}^*} = \emptyset$ and there is no HW-PR-tree that represents $\mathbb{S}^*$, and (D^*, X) otherwise.

Now, for line 8, we need an operation that, for a given constraint S and an HW-PR-tree D^* that contains a node X whose leaf set is $S^* = S \cap L(D^*)$, constructs an HW-PR-tree D such that: $Consistent(D) = \{T : T$ is a tree, V(T) $= L(D^*) \cup S$, $T[S]$ is a path, and $T[L(D^*)] \in Consistent(D^*)\}$. We denote this operation as ***join(D*, X, S**)*** for $S^{**} = S - S^*$, and describe it as follows.

Consider an HW-PR-tree D formed by adding a P-node P to D^* where $Children(P) = S^{**} \cup \{X\}$. Notice that, $T \in Consistent(D)$ iff T is a tree, V(T) $= L(D^*) \cup S$, $T[S]$ is a path, and $T[L(D^*)] \in Consistent(D^*)\}$ (i.e., D is the PR-tree we want). Unfortunately, when $|S^{**}| = 1$ (i.e., $S^{**} = \{u\}$), this operation will not create a valid P-node (since P-nodes require at least three children). So, instead, we create an R′-node Q, where $Children(Q) = \{u, X\}$. Now, if X has an R-node parent $\mathcal{R}$, then we add Q to $\mathcal{R}_{R'}$ and the edge (u, X) to $\mathcal{R}_T$. Otherwise, we create a new R-node $\mathcal{R}$ where $\mathcal{R}_{R'} = \{Q\}$ and $\mathcal{R}_T$ is the edge (u, X). This operation is depicted in fig. 8.

Notice that, `join` creates at most two new nodes each of which have $(|S| + 1)$ children. Furthermore, other than the creation of these nodes, all operations involved in `join` can be performed in constant time. Thus, we have the following result regarding the time complexity of the `join` operation.

Theorem 4.2.1 *For a weak PR-tree D^*, a P-node or R′-node X, and a set S where $S \cap L(D^*) = \emptyset$), `join`($D^*$,$X$,$S$) will require $\mathcal{O}(|S|)$ time to complete.*

With these operations we can now rewrite algorithm 2 as follows:

Algorithm 3 ***Reduction($\mathbb{S}$), version 3:***

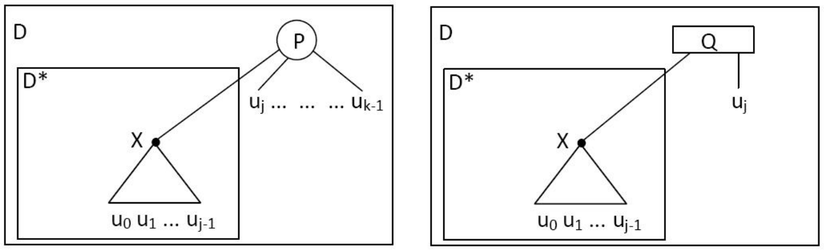

Fig. 8. Illustrating D = join(D^*,X,S^{**}) for an HW-PR-tree D^*, a P-node or R′-node X (where L(X) = $\{u_0, u_1, ..., u_{j-1}\}$), and a set S^{**} = $\{u_j, u_{j+1}, ..., u_{k-1}\}$, where $|S^{**}| > 1$ (left) and $|S^{**}| = 1$ (right).

Solving a connected chordal path-tree problem $\mathbb{S}$ via PR-tree reduction.

```
#Pre: $\mathbb{S}$ = Pre-Proc($G$), for a connected chordal graph $G$.
#Post: return a PR−tree $D$ where $Consistent(D) = \mathcal{T}_G$.
  Initialize $\mathcal{U}_0 = \emptyset$, $D_0$ = ''empty PR-tree''.
  for $i \in [1, n]$: #invariant: $Consistent(D_i) = \mathcal{T}_{\mathbb{S}_i}$
    $\mathcal{U}_i = \mathcal{U}_{i-1} \cup S_{i-1}$.
    $S_i^* = \mathcal{U}_i \cap S_i$.
    $(D_i^*, X_i)$ = reduce($D_{i-1}$, $S_i^*$)
    if $D_i^* == 0$: return ''empty PR-tree'' # since $\mathcal{T}_{\mathbb{S}_i} = \emptyset$.
    $D_i$ = join($D_i^*$, $X_i$, $S_i - S_i^*$)
  return fix_Rnodes($D_n$)
```

4.3 The Reduce Operation

In this section we provide the details of the reduce operation. Recall that, for a given HW-PR-tree D representing a constraint set $\mathbb{S}$ and $S^* \subseteq L(D)$, the primary goal of reduce(D, S^*) is to produce an HW-PR-tree, D^* representing $\mathbb{S} \cup \{S^*\}$. Additionally, this operation must produce a P-node or R′-node X that is the LCA of S^* in D^* such that $L(X) = S^*$. We will refer to D, S^*, D^*, and X throughout this section. We call D ***reducible*** when there is a tree $T \in Consistent(D)$ such that $T[S^*]$ is a path.

Our discussion of reduce(D, S^*) follows similarly to the algorithm for constructing a PQ-tree given in [2]. In particular, we implement the reduce operation through a bottom-up template matching approach on the nodes of D. A template consists of a ***pattern*** and a ***replacement*** represented in terms of a given constraint S^*. We have two types of templates: P-node and weak R-node. The weak R-node templates also make changes to the R′-nodes associated with the weak R-node being processed.

The application of a template to a node N only affects D locally (i.e., only N, N's children, and the weak R-nodes related to N's children are affected). We consider a node to be ***matched*** after a template has been applied to it and we consider a P-node eligible for matching after all of its children have been matched. Furthermore, we consider a node to be ***finished*** after all of its P-node parents have been matched, and we consider a weak R-node eligible for matching after all of its children are finished. This is how we control the traversal of D in a bottom-up manner. In particular, after a node has been matched, its P-node parents will be matched prior to its weak R-node parent. It is important to note that the templates are formed so that, if there is no template that applies to a node that is eligible for matching, D will not be reducible and we will halt the `reduce` operation and return $(0, 0)$.

Due to the lengthy case based nature of the templates, we have placed the templates themselves in an addendum [3].

This discussion of the `reduce` operation is separated into three subsections. The first subsection presents local properties of nodes which imply the irreducibility of D (the templates rely heavily on this discussion). The following subsection consists of an overview of the P-node and weak R-node templates. The final subsection provides an implementation of `reduce`.

4.3.1 Irreducibility and Template Terminology

In this subsection we examine D and its nodes with respect to S^*. During this examination we introduce most of the terminology that is used to describe the templates. We will also observe several properties each of which (when satisfied) imply that D is not reducible.

A leaf u of D is said to be ***full*** when $u \in S^*$; otherwise, u is ***empty***. A node N is said to be ***full*** when all of its leaves are full and N is said to be ***empty*** when all of its leaves are empty. The ***pertinent subtree of D with respect to S****, denoted $Pertinent(D, S^*)$, is the subtree of minimum height whose frontier contains all of S^* (i.e., the subtree "sourced" from the LCSA of S^*, see fig. 9). Notice that, (by corollary 3.5.6) the pertinent subtree is unique. A P-node or R′-node is said to be ***pertinent*** when it belongs to the pertinent subtree and it has a full leaf (note: since LCSAs do not contain R′-nodes, every pertinent R′-node will have at least one parent in the pertinent subtree). A weak R-node $\mathcal{R}$ is ***pertinent*** when at least one of its R′-nodes is pertinent or when an element of the LCSA of S^* is a subpath of $\mathcal{R}_T$.

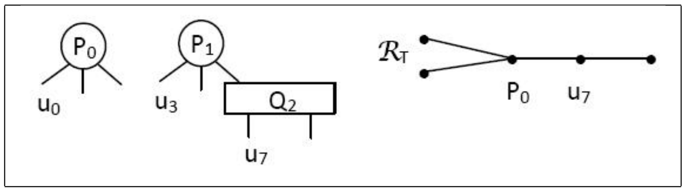

Fig. 9. $Pertinent(D, S^*)$, for D from fig. 7 and $S^* = \{u_0, u_3, u_7\}$.

Additionally, N is said to be ***bridged*** when it has two P-node or weak R-node parents in $Pertinent(D, S^*)$. In this sense we refer to a parent N^* of

N as a ***pertinent parent*** when it is pertinent and is either a P-node or weak R-node (R′-nodes are never considered as pertinent parents). Note: at most one parent of a node can be a weak R-node (by def. 3.1.2). Thus, when N is bridged it either has two pertinent P-node parents and we call N ***p-bridged***, or it has one pertinent P-node parent and one pertinent weak R-node parent and we call N ***r-bridged***. We focus on the case when N has two or fewer such parents in the pertinent subtree since having three or more would indicate that D is not reducible. In particular, each pertinent parent N^* implies the existence of an ancestor of N (i.e., N^* or an ancestor of N^*) with a full leaf that is not a leaf of N. Notice that, in order for D to be reducible, some such full leaf must be adjacent to a leaf of N in some $T \in Consistent(D)$. Furthermore, since D is helly, each pertinent parent will have a distinct such full leaf. More specifically, when N has three (or more) pertinent parents there will be three such full leaves, and, consequently, for every tree $T \in Consistent(D)$ either $T[S^*]$ will not be connected or $T[S^*]$ will not be a path (i.e., D is not reducible).

A pertinent child N of a weak R-node $\mathcal{R}$ is considered ***open*** when it has no pertinent neighbours in $\mathcal{R}_T$; N is considered ***one-sided*** when it has one pertinent neighbour in $\mathcal{R}_T$; N is considered ***closed*** when it has two pertinent neighbours in $\mathcal{R}_T$. We focus on the case when N has two or fewer pertinent neighbours since having three or more would indicate that D is not reducible. In particular, if D contains a weak R-node $\mathcal{R}$ whose pertinent children do not induce a path in $\mathcal{R}_T$ (i.e., the subgraph of $\mathcal{R}_T$ induced by $\mathcal{R}$'s pertinent children is disconnected or has a node whose degree is three or larger), D will not be reducible.

We have similar properties for a full R′-node Q with respect to its "left-most" child C and its "right-most" child C^*. In particular, Q is said to be ***accessible*** when both C and C^* are one-sided. We treat this as the default case for a full R′-node (i.e., unless otherwise stated, a full R′-node is always accessible). Q is said to be ***blocked*** when (w.l.o.g.) C is one-sided and C^* is closed. Finally, when both C and C^* are closed, Q is called ***surrounded***.

Using the above terminology we observe several conditions (similar to those we have already seen) each of which imply that D is not reducible:

(1) A pertinent node with ≥ 3 of any combination of the following: bridged children, blocked-full R′-node children, and pertinent parents.
(2) An r-bridged node that is closed.
(3) A surrounded R′-node with a pertinent parent.

Recall that, the goal of the `reduce` method is the HW-PR-tree D^* where: $T \in Consistent(D^*)$ iff $T \in Consistent(D)$ and $T[S^*]$ is a path. If such a D^* exists, S^* is a good leaf set of D^*. Thus, (by theorem 3.5.8) S^* has a unique LCA in D^* whose leaf set is precisely S^*. In particular, the pertinent subtree of D^* with respect to S^* has a single "source" and every pertinent node is full. Furthermore, there are no bridged nodes in D^* with respect to S^*.

To achieve these properties we prove the following lemma. This lemma is proven in two parts. First with respect to the P-node templates, then with respect to the weak R-node templates (these are proven in the addendum [3]). Together, these two lemmas imply the Template Lemma.

Lemma 4.3.1 ***(Template Lemma)*** *The following invariant is maintained throughout the template matching process:*

> ***General Invariant:***
> *(1) If an eligible node has no applicable template, D is not reducible.*
> *(2) Once a node N of D has been matched (i.e., a template has been applied to N), it will satisfy S^*.*
> *(3) After all of the pertinent P-node parents of a P-node or R′-node N have been matched, N will not be bridged.*
> *(4) After all of the pertinent parents of a P-node or R′-node N have been matched, N only remains pertinent if it is full.*

It is useful to keep this invariant in mind while discussing the templates.

4.3.2 The Templates

Prior to beginning the template matching we traverse the tree bottom-up from the set S^* to locate its LCSA and to identify the pertinent nodes of D. Additionally, we identify whether each pertinent node is p-bridged or r-bridged, and open, one-sided, or closed (note: if we encounter a node with three or more pertinent parents or pertinent neighbours, we halt unsuccessfully indicating that D is not reducible). Template matching is only performed on pertinent nodes.

There is only one template for the leaves (i.e., when a leaf is a member of S^*). This template does not change D. However, the matching will mark the leaf being processed as full.

The situation for non-leaf nodes is not so simple. For a non-leaf node N, we must ensure that N satisfies S^* (i.e., $S^* \cap L(N)$ is a good leaf set of N) while providing N's pertinent parents with the opportunity to satisfy S^*. Additionally, we must make sure that N's children will not be bridged.

We now provide an overview of the P-node templates followed by an overview of the weak R-node templates (note: the weak R-node templates will affect $\mathcal{R}$'s tree and $\mathcal{R}$'s set of R′-nodes).

The P-node templates are presented with respect to a pertinent P-node P (see the addendum [3]). These templates are presented in four groups. In the first group, P is the sole parent of its children and only has full children. The second group expands on the first by allowing P to have bridged-full children and empty children. In the second group, when P has a pertinent parent, P might be replaced by an R′-node with some full and some empty children. We refer to such an R′-node as ***partial***. The third group further introduces blocked-full R′-node children to P. In the fourth group, P is allowed partial children in addition to the possibilities of the third group. The final group completes the options for P's children by including bridged-partial children. Since P can have at most two pertinent parents, we further subdivide each of the latter four groups into three cases regarding the number of pertinent parents of P (i.e., when P has zero, exactly one, or exactly two pertinent parents).

After presenting these groups, we prove the template invariant in the context of P-nodes. In particular, we prove the following invariant:

P-node Invariant:
(1) If an eligible P-node has no applicable template, D is not be reducible.
(2) Once a P-node has been matched, it will satisfy S^*.
(3) After all of a node N's pertinent P-node parents have been matched, N will not be bridged.
(4) After all of a node N's pertinent P-node parents have been matched, N only remains pertinent if it is full or has a pertinent weak R-node parent.

Notice that, by (3) of the P-node Invariant, if we match all of a node's pertinent P-node parents prior to matching its pertinent weak R-node parent we do have to worry about bridged children when matching a weak R-node (i.e., slightly simplifying the templates required for weak R-nodes). This is precisely why our algorithm waits until the children of a weak R-node are finished before processing it. Furthermore, due to the helly property, insisting on a P-node before weak R-node priority when matching the pertinent parents of a bridged node is a valid approach. More specifically, if a pertinent P-node parent P of a bridged node N had to wait for N's pertinent weak R-node parent to be matched before all of P's children could be matched, we would have a violation of the helly property.

We now provide an overview of the weak R-node templates (the templates themselves are in the addendum [3]. Recall that the pertinent children of a weak R-node $\mathcal{R}$ must induce a path in $\mathcal{R}_T$ in order for D to be reducible. We refer to this path as $\mathcal{R}$'s ***pertinent path*** and use $\mathcal{R}_{PP}$ to denote it.

The weak R-node templates are separated into four groups. The first group consists of the case when $\mathcal{R}$'s pertinent path only consists of full children. In the next group we add the possibility that $\mathcal{R}_{PP}$ contains blocked-full R′-nodes. In the third group, $\mathcal{R}_{PP}$ has two partial nodes together with full nodes. The final group considers the case when $\mathcal{R}_{PP}$ has one partial node together with full nodes and blocked-full R′-node(s). Moreover, we separate each of these groups into subcases based on $\mathcal{R}$'s pertinent R′-nodes.

After presenting these groups, we prove the template invariant in the context of weak R-nodes. In particular, we prove the following invariant:

Weak R-node Invariant:

(1) If an eligible weak R node has no applicable template, D is not reducible.

(2) Once a weak R-node $\mathcal{R}$ of D has been matched, $\mathcal{R}$ and every R'-node that was in $\mathcal{R}_{R'}$ prior to replacement will satisfy S^*.

(3) After all of the pertinent P-node parents of a P-node or R'-node N have been matched, N will not be bridged.

(4) After all of the pertinent parents of a P-node or R'-node N have been matched, N only remains pertinent if it is full.

Notice that, the Template lemma (see lemma 4.3.1) follows from the combination of the P-node and weak R-node invariants.

4.3.3 Finalizing the Reduce Operation

In this subsection we present an implementation of the `reduce` operation. This consists of three parts. First, we discuss the process in which we identify the pertinent subtree (i.e., how to locate the LCSA of S^* in D and mark the pertinent nodes with their relevant properties). We then present the final version of the `reduce` operation. We finish this subsection by discussing how to locate the LCA of S^* once the template matching has finished.

To identify the pertinent subtree we use an operation called ***`label(D,S*)`***. In particular, this operation identifies the LCSA of S^* in D, the pertinent nodes of D, and whether each pertinent node is:

- Open, one-sided, or closed.
- Accessible, blocked, or surrounded.
- An element of the $LCSA(D, S^*)$, p-bridged, r-bridged, or otherwise (i.e., has a single pertinent parent).

This method first traverses D from S^* (i.e., bottom-up) marking each node it encounters ***visited*** (i.e., every node in D with a leaf in S^* becomes visited). It also keeps track of the number of visited children of every visited node. Then, it traverses D from D's visited sources (i.e., top-down). On the way down the "highest" P-nodes and weak R-nodes with multiple visited children form the LCSA of S^* in D and are the first nodes to be identified as pertinent (and are marked as such). Continuing down D it records the appropriate properties of each pertinent node. Notice that, this method runs in time proportional to the number of visited nodes (i.e., the number of nodes with leaves in S^*).

We now present the `reduce` operation (see algorithm 4). Recall that a node N is: ***matched*** once a template has been applied to it, and ***finished*** after all of its pertinent P-node parents have been matched (i.e., when N is not bridged). In this sense we do not enqueue a P-node until all of its pertinent children are matched and we do not enqueue a weak R-node until all of its pertinent children are finished.

Algorithm 4 ***reduce(D, S^*)***
Reducing an HW-PR-tree by a given constraint [10] .

```
#Pre: $D$ is an HW-PR-tree representing a constraint set $\mathbb{S}$, and $S^* \subseteq L(D)$.
#Post: $D^*$ is an HW-PR-tree representing $\mathbb{S} \cup \{S^*\}$ and $X = LCA(D^*, S^*)$.
  label$(D, S^*)$
  Initialize $\mathcal{Q}$ as a queue containing $S^*$.
  while $\mathcal{Q}$ is not empty:
    $N$ = $\mathcal{Q}$.dequeue() # remove and return the first element of $\mathcal{Q}$.
    Initialize $newlyMatched = \emptyset$, $newlyfinished = \emptyset$.
    if a P-node or leaf template applies to $N$:
      Substitute the replacement for the pattern in $D$.
      Set $newlyMatched$ = $\{N\}$.
      Set $newlyFinished$ as $N$'s children which are no longer bridged.
    elif a weak R-node template applies to $N$:
      Substitute the replacement for the pattern in $D$.
      Set $newlyMatched$ to be the pertinent R$'$-nodes in $N_{R'}$.
    else: return (0,0) #$D$ is not reducible by $S^*$.
    for each node $N^*$ in $newlyMatched$:
      Mark $N^*$ as matched.
      if $N^*$ is not bridged: Add $N^*$ to $newlyFinished$.
      for each pertinent P-node parent $Y$ of $N^*$:
        if $Y$'s pertinent children are all matched:
          Add $Y$ to the end of $\mathcal{Q}$.
    for each node $N^*$ in $newlyFinished$:
      Mark $N^*$ as finished.
      if $N^*$ has a pertinent weak R-node parent $\mathcal{R}$ and every pertinent
           child of $\mathcal{R}$ is finished: Add $\mathcal{R}$ to the end of $\mathcal{Q}$.
  $D^* = D$.
  if $|LCSA(D^*, S^*)| > 1$: return $\emptyset$
  $X = LCA(D^*, S^*)$
  if $X$ is a path in a weak R-node $\mathcal{R}$:
    Set $X$ as the R$'$-node corresponding to $\mathcal{R}_{PP}$.
    # note: when no such R$'-$node exists, we create it.
  return ($D^*$, $X$)
```

Note: the correctness and complexity of the template matching portion are discussed briefly in sections 5 and 6 respectively, and in detail in [3].

We now discuss how to locate the LCA of S^* after the template matching has completed successfully (i.e., how to determine $LCA(D^*, S^*)$). Notice that, through the template matching process some of the pertinent nodes will lose their pertinent status. In particular, it is easy for us to keep track of which nodes remain pertinent as we apply templates. Thus, we can sim-

[10] This method proceeds similarly to *S-reductions* from [2].

ply traverse D^* from S^* (i.e., bottom-up) along the pertinent nodes in order to locate S^*'s LCSA. Additionally, by the Template Lemma (see lemma 4.3.1), only full nodes will remain pertinent after the template matching has finished. Furthermore, no node in D^* will be bridged with respect to S^*. Thus, for any pair X, X^* of pertinent nodes with no pertinent parents (i.e., elements of $LCSA(D^*, S^*)$, X and X^* will have no leaves in common (i.e., $L(X) \cap L(X^*) = \emptyset$). In particular, the elements of the LCSA of S^* will partition S^* into disjoint subsets (i.e., each subset will be the leaf set of an element of the LCSA). More specifically, there is no node N in D^* whose leaf set intersects the leaf sets of both X and X^*. Therefore, there is no edge which connects a leaf of X to a leaf of X^* in D^*'s frontier. Moreover, there is no such edge in the frontier of any HW-PR-tree equivalent to D (since such an edge would have to come from a node which shared a leaf with each of X and X^*). With this in mind, we will halt unsuccessfully when the LCSA of S^* in D^* has more than one element. Additionally, when the LCSA has exactly one element we will set X to be that element and return it along with D^*. This leads to the following two observations which will be useful for our discussion of total correctness in section 5.

Observation 4.3.3.1 *For an HW-PR-tree D and* $S^* \subseteq L(D)$, *if the template matching portion of the (reduce) completes successfully (i.e., converting D to* D^**), then* S^* *will induce a disconnected collection of paths in every* $T \in Consistent(D^*)$.

Observation 4.3.3.2 *For an HW-PR-tree D and* $S^* \subseteq L(D)$, *if (reduce) completes successfully and returns* (D^*, X), *then* S^* *will be a good leaf set of* D^* *and* X *will be the LCA of* S^* *in* D^*.

Notice that, to locate the LCSA of S^* we will only traverse the pertinent nodes in D^*. In particular, we will traverse a collection of disjoint subtrees of D^* (i.e., one for each element of the LCSA). Furthermore, since every element of the LCSA is full, every node in every subtree will be full. Therefore, the total number of nodes that we will traverse is proportional to the number of elements in S^*). This provides the following observation regarding the runtime required to locate the LCSA of S^*.

Observation 4.3.3.3 *To locate the LCSA of* S^* *in* D^* *during the execution of* `reduce`*(D,*S^**) on an HW-PR-tree D and* $S^* \subseteq L(D)$ *takes time proportional to the size of* S^*. *In particular, the runtime is:* $\mathcal{O}(|S^*|)$.

4.4 Fixing the Weak R-nodes

In this section we discuss the final component of our algorithm. Recall that, the `Reduction` algorithm (see algorithm 3 on pg. 33) iteratively applies `reduce` and `join` operations to produce an HW-PR-tree D_n where $Consistent(D_n)$ is precisely $\mathcal{T}_{\mathbb{S}}$. However, such an HW-PR-tree is not sufficient to determine if $\mathcal{T}_{\mathbb{S}}$ is not empty (i.e., if the graph G that $\mathbb{S}$ was generated from is a path graph). In particular, $Consistent(D_n)$ might be empty. Thus, to ensure that $\mathcal{T}_{\mathbb{S}} \neq \emptyset$, we attempt to find a PR-tree D which is equivalent to D_n. Recall that the elements of $Consistent(D_n)$ are precisely the frontiers of the PR-trees which are equivalent to D_n. Thus, such a PR-tree exists iff $Consistent(D_n) \neq \emptyset$. Furthermore, the difference between a PR-tree and an HW-PR-tree is that the HW-PR-tree can have weak R-nodes, but the PR-tree must have R-nodes. Thus, if we can apply an equivalence transformation ψ to D_n so that $\psi(D_n)$'s weak R-nodes are R-nodes, we will have "fixed the weak R-nodes" of D_n. With this in mind we present the ***fix_Rnodes*** which takes an HW-PR-tree (i.e., D_n) as input and (when possible) returns an equivalent PR-tree.

Recall that, a weak R-node is an R-node when it is PWF (see def. 3.2.6). We note the following two observations regarding this definition:

(1) Given an R-node $\mathcal{R}$, if we apply $\eta^{reverse}$ to every R′-node in $\mathcal{R}_{R'}$ (i.e., reversing the order of every R′-node in $\mathcal{R}_{R'}$), the result is an R-node (i.e., PWF is preserved when reversing every element in $\mathcal{R}_{R'}$).

(2) Consider a weak R-node $\mathcal{R}$ with $N_0, N_1 \in \mathcal{R}_{R'}$, and $X_0, X_1 \in Children(\mathcal{R})$ such that N_0 and N_1 together with X_0 and X_1 demonstrate that $\mathcal{R}$ is not PWF (i.e., as above). Let X_1^* be the child of $\mathcal{R}$ which is adjacent to X_0 on the path in $\mathcal{R}_T$ from X_0 to X_1 (i.e., when X_0 and X_1 are adjacent, $X_1^* = X_1$). Thus, X_1^* is also a child of both N_0 and N_1. In particular, N_0 and N_1 together with X_0 and X_1^* also demonstrate that $\mathcal{R}$ is not PWF. More specifically, when $\mathcal{R}$ is not PWF, the violation will occur with respect to an edge X_0X_1 of $\mathcal{R}_T$.

With the second observation this in mind, we refer to each edge of $\mathcal{R}_T$ as either ***unpaired*** or ***paired***, where an edge is unpaired when both of its incident nodes are leaves, and an edge is paired otherwise.

These two observations lead to the following forcing algorithm that transforms a weak R-node into an R-node (when possible). In particular, we start with an arbitrary R′-node $Q \in \mathcal{R}_{R'}$, mark Q as ***ordered***, and orient its paired edges in $\mathcal{R}_T$ according to the order of Q's children. Notice that, by observation (1) above, if we can make $\mathcal{R}$ PWF then we can do so with Q's children ordered in this way. We now consider an unordered R′-node Q^* that contains a newly oriented paired edge e. This oriented edge forces the ordering of Q^*'s children. In particular, we first reverse Q^*'s ordering (as needed) so that the order on its children matches e. We then mark Q^* as ***ordered*** and orient Q^*'s unoriented paired edges in $\mathcal{R}_T$ according to the order of Q^*'s children. We continue this process for every such Q^* (i.e., until we exhaust any newly oriented edges). If we encounter a Q^* which has two paired edges e and e^* that are already oriented, but are incompatible (i.e., neither order on Q^*'s children satisfies both edges), we know that we cannot make $\mathcal{R}$ PWF. More specifically, the orientation of each edge e and e^* was forced and this forcing can be traced back to Q. This process orders every R′-node reachable from Q through a sequence of shared paired edges. However, this will not necessarily order all the R′-nodes in $\mathcal{R}_{R'}$. In particular, we may run out of forcing edges before we have exhausted $\mathcal{R}_{R'}$. Fortunately, since we have exhausted all of the forcing edges, we can safely restart the forcing process by picking a new unordered Q from $\mathcal{R}_{R'}$ and continuing as before. Notice that, if we never encounter an R′-node with a pair of incompatible edges, this approach will make $\mathcal{R}$ PWF since it will have ordered every R′-node of $\mathcal{R}$ and ensured its order is consistent with all previously ordered R′-nodes.

We call this operation ***makePWF*** and it will take a weak R-node $\mathcal{R}$ as input and (when possible) apply appropriate equivalence transformations to the elements of $\mathcal{R}_{R'}$ so that $\mathcal{R}$ will become PWF. Notice that, if we wanted to convert a weak R-node into a strong R-node (i.e., to make it pairwise consistent (PC)), we can use the same algorithm except we consider every edge of $\mathcal{R}_T$ to be paired. We call this operation ***makePC***.

Notice that, we can implement *makePWF*($\mathcal{R}$) so that it "orders" each R′-node Q in $\mathcal{R}_{R'}$ exactly once and, when it does, it examines each edge of $\mathcal{R}_T[Children(Q)]$ exactly once. Thus, *makePWF*($\mathcal{R}$) completes in time proportional to the sum of the sizes of the R′-nodes in $\mathcal{R}_{R'}$. Similarly, *makePC*($\mathcal{R}$) has the same timing characteristics.

Using this method we can easily implement the *fix_Rnodes* operation. In particular, we apply *makePWF* to every weak R-node in D (note: applying *makePWF* to a weak R-node in D does not affect any other weak R-node in D since, by definition, no pair of weak R-nodes have R′-nodes in common). This method's correctness and runtime complexity follows directly from that of *makePWF*. Similarly, we can create an operation ***strengthen_Rnodes*** (using *makePC*) which takes a weak PR-tree and converts it into an equivalent strong PR-tree (when possible). Thus, we have the following results:

Theorem 4.4.1 *For an HW-PR-tree D, fix_Rnodes(D) results in a PR-tree equivalent to D or $\emptyset$ when no such PR-tree exists, and strengthen_Rnodes(D) results in a strong PR-tree equivalent to D or $\emptyset$ when no such strong PR-tree exists.*

Theorem 4.4.2 *For an HW-PR-tree D, both strengthen_Rnodes(D) and fix_Rnodes(D) halt in time proportional to the size of the R′-nodes in D; i.e.:*
$\mathcal{O}$($\sum\{$ # of edges in $(Q_R)_T[Children(Q)]$: for every R′-node Q of D $\}$)

5 The Correctness of PR-tree Construction

In this section we demonstrate the correctness of our PR-tree construction algorithm (i.e., the *Reduction* method). An implementation of *Reduction* is presented as algorithm 3 on pg. 33. Recall that this method takes a constraint set $\mathbb{S} = \{S_0, S_1, ..., S_{n-1}\}$ built from a connected chordal graph

$G = (V, E)$ (i.e. $\mathbb{S}$ = `PreProc`(G) [11]) and should produce a PR-tree D where: $Consistent(D) = \mathcal{T}_G$ (i.e., the consistent set of D is precisely the path-tree models of G). This is presented as theorem 5.1 and is followed by a short discussion of constructing strong PR-trees and rooted PR-trees.

Theorem 5.1 *For a graph G,* `Reduction`*(*`PreProc`*(G)) will produce a non-empty PR-tree D where $Consistent(D) = \mathcal{T}_G$, iff G is a path graph.*

Proof:
We first prove that the `Reduction` method is correct given that `reduce` is correct (i.e., as in (1)), then we prove that `reduce` is correct (see (2) below).

(1): If `reduce` is correct, then `Reduction` is correct.

proof: We first prove an invariant of `Reduction`'s main loop; i.e.,
(Invariant) D_i is an HW-PR-tree where: $Consistent(D_i) = \mathcal{T}_i$.

Notice that the `Reduction` process starts with an empty PR-tree D_0, and $\mathcal{T}_0 = \emptyset$ (i.e., $Consistent(D_0) = \mathcal{T}_0$). Thus, the base case is satisfied.

We now consider the i^{th} iteration of this loop (i.e., we assume that D_{i-1} satisfies the invariant and demonstrate that D_i also satisfies the invariant).

The first operation that is performed is to reduce D_{i-1} by the subset S_i^* of the constraint S_i which is contained in the leaf set of D_{i-1}. Notice that, S_i^* will not be empty due to the ordering of the constraints. Furthermore, by the correctness of `reduce` (see (2) below), `reduce`$(D_{i-1}$,$S_i^*)$ will return (D_i^*, X_i) where: $T \in Consistent(D_i^*)$ iff $Consistent(D_{i-1})$ and $T[S_i^*]$ is a path (i.e., $Consistent(D_i^*) = \mathcal{T}_{\mathbb{S}_{i-1} \cup \{S_i^*\}}$). Recall that, by observation 4.1.1, $\mathcal{T}_{\mathbb{S}_i}$ $= \{T : T$ is a tree, $V(T) = \mathcal{U}_i$, $T[\mathcal{U}_{i-1}] \in \mathcal{T}_{\mathbb{S}_{i-1} \cup \{S_i^*\}}$, and $T[S_i]$ is a path$\}$. Notice that, $\mathcal{T}_{\mathbb{S}_{i-1} \cup \{S_i^*\}} = \emptyset$ when $D_i^* = 0$. Thus, when `reduce`(D_{i-1}, S_i^*) returns $(0, 0)$, we correctly return $\emptyset$ from the `Reduction` method since $\mathcal{T}_{\mathbb{S}}$ will be empty. Now, when $D_i^* \neq \emptyset$, we apply `join` on $(D_i^*, X_i, S_i - S_i^*)$ to produce D_i. Recall that, `join` adds a new P-node [12] to D_i^* whose children are $\{X_i\} \cup (S_i - S_i^*)$. The addition of this new source node does not affect the helly status of this weak PR-tree (i.e., if D_i^* is helly, so is D_i). Thus, D_i is an HW-PR-tree where $Consistent(D_i)$ is $\{T : T$ is a tree, $V(T) = \mathcal{U}_i$, $T[\mathcal{U}_{i-1}] \in Consistent(D_i^*)$, and $T[S_i]$ is a path$\}$ (i.e., $Consistent(D_i) = \mathcal{T}_i$).

We have now shown that the loop invariant given above is true. Therefore, when the loop completes we have an HW-PR-tree D_n such that $Consistent(D_n) = \mathcal{T}_n$ (i.e., $Consistent(D_n) = \mathcal{T}_{\mathbb{S}}$). Now all that remains is to convert D_n into a PR-tree (when possible). As we have seen in theorem 4.4.1, `fix_Rnodes` accomplishes this. Notice that, since we have produced a PR-tree, $Consistent(D)$ is not empty (i.e., $Frontier(D) \in Consistent(D)$).

[11] $\mathbb{S} = \mathbb{S}_G$, where $S_i = S_{v_i}$ for $V = \{v_0, v_1, ..., v_{n-1}\}$ and $\mathbb{S}$ is ordered so that $\forall i \in [1, n-1], \exists j \in [0, i-1]$ such that $S_i \cap S_j \neq \emptyset$.
[12] The degenerate cases (i.e., when $|S_i - S_i^*|$ is one or zero) also work as needed.

Furthermore, if the `Reduction` method exits prior to returning a PR-tree, it returns $\emptyset$. Therefore, pending the correctness of `reduce`, the `Reduction` method is correct (i.e., completing the proof of (1)). $\square$

(2): Given an HW-PR-tree D and $S^* \subseteq L(D)$, if $(D^*, X) = \texttt{reduce}(D, S^*)$ then D^* is an HW-PR-tree such that $T \in Consistent(D^*)$ iff $T \in Consistent(D)$ and $T[S^*]$ is a path.

proof: We split the proof of (2) into three parts. We first demonstrate that the template matching process maintains the helly property. The next step is to prove that $T \in Consistent(D)$ and $T[S^*]$ is a path when $T \in Consistent(D^*)$. The final part proves that $T \in Consistent(D^*)$ when $T \in Consistent(D)$ and $T[S^*]$ is a path.

(2a): The helly property is maintained by the template matching process.

proof: Recall that a weak PR-tree D is helly when every loop $\{N_0, N_1, ..., N_{k-1}\}$ [13] of nodes, has a non-empty common leafset $L = \bigcap_{i=0}^{k-1} L(N_i)$, and L has an LCA Z where $L(Z) = L$. We now consider the nodes created during the template matching process. Notice that, the nodes created by P-node templates are inserted in between the P-node P being matched and its children. Clearly, these nodes do not cause a violation to the helly property (since such a violation would imply that P would be involved in a loop which violated the helly property). Furthermore, no P-node template changes the leaf set of any P-node or R′-node. The weak R-node templates do not create any new P-nodes or R′-nodes and they do not change the leaf set of any P-node or R′-node. Therefore, the helly property is maintained by the application of any template; i.e., D^* is helly. $\diamond$

(2b): If $T \in Consistent(D^*)$, then $T \in Consistent(D)$ and $T[S^*]$ is a path

proof: Consider $T \in Consistent(D^*)$. If such a tree exists, D must have been successfully reduced (otherwise, D^* would be $\emptyset$). By definition D^* has an equivalent PR-tree D^{**} whose frontier is T. The basic idea of this proof is to construct a PR-tree D^{***} equivalent to D, where $Frontier(D^{***})$ is T. To accomplish this we will "undo" the template matching which reduced D, but without undoing the equivalence transformations which were used during the template matching. Recall that the process of applying a template has two stages: the pattern must be matched (possibly requiring an equivalence transformation), then the replacement is made. Consider the sequence of template applications used in $\texttt{reduce}(D, S^*)$ in reverse order. If we undo the replacement but not the equivalence transformation, the template matching process can be run in reverse. Notice that, the result of applying this reversed template-matching to the PR-tree D^{**} is a PR-tree D^{***} which is equivalent to D, but has the same

[13] A loop requires two conditions: (1) $N_i \notin Desc(N_j)\ \forall i, j \in [0, k-1]$; and (2) $N_i \cap N_{i+1} \neq \emptyset$ for every $i \in [0, k-2]$ and $L(Nk-1) \cap L(N_0) \neq \emptyset$).

frontier as D^{**} (i.e., $T \in Consistent(D)$. Furthermore, by observation 4.3.3.2, S^* is a good leaf set of D; i.e., $T[S^*]$ is a path. Thus proving (2b). $\diamond$

(2c): If $T \in Consistent(D)$ and $T[S^*]$ is a path, then $T \in Consistent(D^*)$.

proof: Since $T \in Consistent(D)$, there is a PR-tree equivalent to D whose frontier is T. Let D^{**} be such a PR-tree. We will now apply `reduce` to D^{**} with respect to S^*. The fact that S^* is a path in T leads to the following properties of the pertinent subtree of D^{**}. Notice that no pertinent node will have three or more pertinent parents. With this condition in mind we examine the frontier of a pertinent node N.

(i) If N is a bridged P-node or R′-node, then either:
- All of N's leaves are in S^* and the frontier of N is a subpath of $T[S^*]$ whose elements all have degree two (i.e., even the endpoints of N's frontier will have degree two in $T[S^*]$); or
- Exactly one of N's leaves is in S^*, this leaf is an endpoint of the frontier of N, and it has degree two in $T[S^*]$.

(ii) If N is a P-node or R′-node with exactly one pertinent parent, then:
- A full leaf of N will be an endpoint of N's frontier and has at most one neighbour in $T[S^*]$ which is not a leaf of N; and
- N's full leaves will form a same subpath of N's frontier and $T[S^*]$.

(iii) If N is a weak R-node or a P-node [14] with no pertinent parents, then N's frontier contains a full subpath which is a subpath of $T[S^*]$ (i.e., the endpoints of this subpath may each have up to one full neighbour outside of N's frontier, but the internal nodes cannot have any such neighbours).

Thus, no pertinent node with more than one full leaf will have three or more of any combination of the following: pertinent parents, partial children, bridged descendents, and blocked-full descendents. Additionally, no pertinent node with exactly one full leaf can have either three or more pertinent parents, or two pertinent parents and a bridged descendent.

We call a pertinent node ***occluded*** when it has an empty, bridged, partial, or blocked-full child. Notice that, after any occluded P-node or R′-node with a pertinent parent is matched it either becomes a blocked-full R′-node or a partial R′-node. In the latter case, its sequence of children, examined (w.l.o.g.) from left to right is a sequence of full nodes followed by a sequence of empty nodes where:
- The "left-most" full child is either one-sided or open; and
- Each "middle" child is closed and either a P-node or an accessible R′-node; and
- The "right-most" full child is closed and not bridged, or optionally bridged and either one-sided or open.

Furthermore, since T is a tree, D^{**} is a PR-tree (i.e., every weak R-node in

[14] Recall that pertinent R′-nodes must have at least one pertinent parent.

D^{**} is an R-node). In particular, this means that the moving of relevant elements of each matched weak R-node of D^{**} always succeeds.

These observations guarantee that each pertinent node matches one of the templates and the replacements can be performed successfully. Thus, we now consider D^{***} to be D^{**} after successfully completing the template matching process.

Notice that, since T is a tree and $T[S^*]$ is a path, the replacement of each template we apply can be made without the need to use equivalence transformations (i.e., without needing to change the frontier of D^{**}). In particular, T is the frontier of D^{***}. Furthermore, since $T[S^*]$ is a path, by observation 4.3.3.1, S^* is a good leaf set of D^{***}. Thus, `reduce` completes successfully on D^{**} and S^*. Moreover, $\texttt{reduce}(D^{**}, S^*)$ returns (D^{***}, X^{***}) where $T \in Consistent(D^{***})$. Therefore, since D^{**} and D are equivalent and template matching preserves equivalence, T is also in $Consistent(D^*)$, where $(D^*, X) = \texttt{reduce}(D, S^*)$. ◇

Therefore, by (2a), (2b), and (2c), (2) is now proven. □
QED.

We now consider modifying `Reduction` to use `strengthen_Rnodes` in place of `fix_Rnodes` (we call this new method ***`StrongReduction`***). The theorem below follows from the proof above and the correctness of `strengthen_Rnodes`.

Theorem 5.2 *For a graph G,* `StrongReduction(PreProc(G))` *produces a non-empty strong PR-tree D where* $\overrightarrow{Consistent}(D) = \overrightarrow{\mathcal{T}}_G$ *iff G is a directed path graph.*

Furthermore, we can easily use `Reduction` to construct a PR-tree to represent the path-path models of G. Recall that, $\mathcal{P}_G$ consists only of paths. We can easily force every element in a PR-tree's consistent set to be a path by making $L(D)$ a good leaf set of D (i.e, using $L(D)$ as a constraint). In particular, by using $\mathbb{S} = \{S, S_0, S_1, ..., S_{n-1}\}$ where $\{S_0, S_1, ..., S_{n-1}\}$ = `PreProc(G)` and $S = \bigcup_{i=0}^{n-1} S_i$ (i.e., S contains every maximal clique of G and is the ***first*** constraint of $\mathbb{S}$). Thus, if $D = \texttt{Reduction}(\mathbb{S})$, then D represents $\mathbb{S}$. More specifically, $Consistent(D)$ is the set of paths satisfying $\mathbb{S}_G$ (i.e., $Consistent(D) = \mathcal{P}_G$). The structure of this PR-tree is very restricted. In particular, since $L(D)$ is a good leaf set in D, its least common ancestor X has $L(X) = L(D)$. Thus, D is a rooted PR-tree (i.e., D is a PQ-tree by observation 3.4.2). This follows from the fact that `Reduction` only creates source nodes when applying a `join` operation (i.e., by observation 6.1.1). Thus, as in the below theorem, we have re-proven that the PQ-tree (i.e., rooted PR-tree) solves the path-path problem and shown that our algorithm can construct a PQ-tree.

Theorem 5.3 *For a graph G,* `Reduction(`$\{S\} \cup$ `PreProc(G))` *produces a non-empty rooted PR-tree D where* $Consistent(D) = \mathcal{P}_G$ *iff G is an interval graph.*

6 Runtime Complexity of PR-tree Construction

In this section we discuss the runtime complexity of the `Reduction` method. We first examine a PR-tree D produced by this process. In particular, we prove that the "size" of D is linear with respect to the size of the input graph G (see section 6.1). We then use this result to demonstrate that the total running time of `Reduction`(`PreProc`(G)) is $\mathcal{O}(a(n+m) * n * m)$ where $a(s)$ is the inverse Ackermann function from Union-Find (see section 6.2). Note: the details of our usage of Union-Find are given in the addendum [3].

Notice that, the size of the set $\mathbb{S}$ produced by `PreProc`(G) is $\Sigma_{v \in V(G)}|S_v|$. In particular, this is the number of ones in the vertex to maximal clique incidence matrix of G. Furthermore, by the following theorem from [8] (see theorem 6 below), we can relate the size of a PR-tree produced by our algorithm and the running time of our algorithm to the size of the input graph.

Theorem 6 *[8] For a chordal graph, the number of ones in the vertex to maximal clique incidence matrix is $\mathcal{O}(n+m)$.*

6.1 The Constructed PR-tree

Throughout our discussion of the templates (see [3]) we are careful to ensure that no new P-node or R′-node sources are introduced by the application of a template. Thus, the only time we create new source nodes during the `Reduction` process is when we perform the `join` operation (note: in the case of a degenerate join we may still create a new node, namely X_i the LCA of S_i^*). We formalize this in the following observation:

Observation 6.1.1 *For a chordal graph G with constraint set $\mathbb{S} = \{S_0, S_1, ..., S_{n-1}\} =$ `PreProc (G)`, each HW-PR-tree $D_i = \{\{A_0, ..., A_{\alpha-1}\}, \{R_0, ..., R_{\gamma-1}\}\}$ produced during the execution of `Reduction`($\mathbb{S}$) will satisfy the following:*

- *D_i has at most i source nodes which are not weak R-nodes (i.e., $\alpha \leq i$); and*
- *Each such source is the LCA of a distinct constraint (i.e., w.l.o.g. $L(A_j) = S_j$ for $j \in [0, \alpha - 1]$).*

Furthermore, since each P-node has at least three children and each R′-node has at least two children, the number of P-nodes and R′-nodes in the subtree rooted at a any node N is at most the number of leaves of N. Therefore, we can bound on the number of P-nodes and R′-nodes in an HW-PR-tree built during the `Reduction` method (see observation 6.1.2).

Observation 6.1.2 *For a chordal graph G with $\mathbb{S} =$ `PreProc`(G):*
Each HW-PR-tree D_i produced during the execution of `Reduction`($\mathbb{S}$) has at most $2 * \sum_{j=0}^{i} |S_j|$ edges (i.e., parent to child connections) between P-nodes, R′-nodes, and leaves.

We now consider the size of weak R-nodes in a weak PR-tree. Notice that, in a weak PR-tree, each weak R-node has a distinct set of R′-nodes and a distinct set of children. In particular, the number of weak R-node to child edges is bounded by the number of R′-node to child edges, and the number of R′-node to weak R-node connections is bounded by the number of R′-nodes. Furthermore, the size of the tree associated with a weak R-node is proportional to the number of children of that weak R-node. Additionally, for the convenience of `fix_Rnodes`, we will keep track of the R′-nodes associated with each edge of $\mathcal{R}_T$ for each weak R-node $\mathcal{R}$ (note: this is again proportional to the number of R′-node to child edges). With this in mind we can see that the weak R-nodes of a weak PR-tree have a space requirement which is proportional to that of the R′-nodes.

The previous set of observations leads to the following theorem regarding the size of the PR-tree produced by our construction algorithm relative to the size of the input graph.

Theorem 6.1.3 *For a graph G, and a PR-tree D = `Reduction`($\mathbb{S}$) (for $\mathbb{S}$ = `PreProc`(G)), the size of D, denoted $|\boldsymbol{D}|$, will be $\mathcal{O}(\sum_{v \in V(G)} |S_v|)$ (i.e. $\mathcal{O}(n+m)$ by theorem 6) where D will be $\emptyset$ when G is not a path graph.*

6.2 Runtime Complexity

In this subsection we discuss the runtime complexity of the `Reduction` algorithm. We separate this analysis into two parts: the analysis the `reduce` method, and everything else.

The discussion of the `reduce` method relies on the details of the templates and as such is presented in the addendum [3]. In particular, to perform the `reduce` operation use the Union-Find data structure of Tarjan [22] to represent the weak R-nodes of in the current HW-PR-tree (this is described in detail in [3]). In short, we use two instances of Union-Find in order to represent the two families of disjoint sets associated with the weak R-nodes of an HW-PR-tree. More specifically, for an HW-PR-tree D we have $\{Children(\mathcal{R})$: for every weak R-node $\mathcal{R}$ in $D\}$ and $\{\mathcal{R}_{R'}$: for every weak R-node $\mathcal{R}$ in $D\}$ which are each represented by an instance of Union-Find.

Using these data structures, we have shown that `reduce`(D_{i-1},S^*_i) can be performed in $\mathcal{O}(a(\sum_{j=0}^{i} |S_j|) * \sum_{j=0}^{i} |S_j|)$ time (where $a(s)$ is the inverse of Ackermann's function arising from Union-Find) in the context of `Reduction`($\mathbb{S}$) (where $\mathbb{S}$ = `PreProc`(G) for a graph G). Using this result we can see that the total time required by the calls to `reduce` is:
$\mathcal{O}(\sum_{i=0}^{n-1} a(\sum_{j=0}^{i} |S_j|) * \sum_{j=0}^{i} |S_j|)$; i.e., $\mathcal{O}(a(n+m) * n * m)$(by theorem 6).

We complete the analysis of the `Reduction` algorithm by showing that the other operations complete in $\mathcal{O}(n+m)$ time (see theorem 6.2.1 below).

Theorem 6.2.1 *For a graph G, all operations performed during `Reduction`(`PreProc`(G)) **other than** the calls to the `reduce` method can be completed in $\mathcal{O}(n+m)$ time.*

Proof:
Recall that `PreProc`(G) uses $\mathcal{O}(n+m)$ time (see section 4.1). Furthermore, by theorem 4.4.2, the runtime of `fix_Rnodes` on an HW-PR-tree D is:
$\mathcal{O}(\ \sum\{$ # of edges in $(Q_R)_T[Children(Q)]$: for every R′-node Q of D }).

Thus, by observation 6.1.2, the total size of all R′-nodes in the HW-PR-tree (i.e., D_n) on which `Reduction` performs `fix_Rnodes`, is $\mathcal{O}(n+m)$. Thus, the call to `fix_Rnodes` is performed in $\mathcal{O}(n+m)$ time.

We now examine the main loop of the `Reduction` algorithm. Notice that, each iteration of this loop involves a set intersection (i.e. determining $S_i^* = \mathcal{U}_{i-1} \cap S_i$), the execution of `reduce`($D_{i-1}$,$S_i^*$), and the execution of `join`(D_i^*,X_i,$S_i - S_i^*$). For now we set aside the time required for each call to the `reduce` method. Notice that, the set intersection can be performed in $\mathcal{O}(|S_i|)$ time. Similarly, by theorem 4.2.1, the `join` is performed in $\mathcal{O}(|S_i|)$ time. In particular, if we disregard the calls to `reduce` the time used by the main loop is $\mathcal{O}(\sum_{i=0}^{n-1}|S_i|)$ time; i.e., by theorem 6, $\mathcal{O}(n+m)$ time.
QED.

Notice that, the timing of the sequence of `reduce` operations dominates the `Reduction` and `StrongReduction` methods. Therefore, both can be performed in $\mathcal{O}(a(n+m)*n*m)$ time (as in the theorems below).

Theorem 6.2.2 *For a graph G, computing $\mathbb{S}$ =`PreProc`(G), `Reduction`($\mathbb{S}$), and `StrongReduction`($\mathbb{S}$) completes in $\mathcal{O}(a(n+m)*n*m)$ time.*

Note: $a(s)$ is the inverse of Ackermann's function arising from Union-Find [22].

The above theorem together with the correctness of `Reduction` 5.1) and `StrongReduction` (theorems 5.1 5.2) provides the following theorems.

Theorem 6.2.3 ***(PR-tree)*** *For a graph G, a PR-tree D where $Consistent(D) = \mathcal{T}_G$, can be produced in $\mathcal{O}(a(n+m)*n*m)$ time. Additionally, $D \neq \emptyset$ iff $\mathcal{T}_G \neq \emptyset$ (i.e., G is a path graph iff $D \neq \emptyset$).*

Theorem 6.2.4 ***(Strong PR-tree)*** *For a graph G, a strong PR-tree D where $\overrightarrow{Consistent}(D) = \overrightarrow{\mathcal{T}}_G$, can be produced in $\mathcal{O}(a(n+m)*n*m)$ time. Additionally, $D \neq \emptyset$ iff $\overrightarrow{\mathcal{T}}_G \neq \emptyset$ (i.e., G is a directed path graph iff $D \neq \emptyset$).*

The PR-tree and Strong PR-tree theorems complete our discussion of PR-tree construction. In particular, they demonstrate that the PR-tree and strong PR-tree solve the path-tree and directed path-tree problems respectively.

7 Concluding Remarks

The main contribution of this work is the ***PR-tree*** (see def. 3.2.11): a new data structure which characterizes the path-tree models of graphs (see theorem 6.2.3) and (with minor changes) also characterizes directed path-tree and path-path models of graphs (see theorems 6.2.4 and 5.3 respectively).

In particular, for a graph G, there exists a non-empty PR-tree D whose consistent set is precisely the G's path-tree models (i.e., $\mathcal{T}_G$) iff G is a path graph. Additionally, we have demonstrated an algorithm (see algorithm 3) which, in $\mathcal{O}(a(n+m) * n * m)$ time, not only constructs such a PR-tree, but also guarantees that its size is linear with respect to the size of G (see theorem 6.1.3). An additional consequence of this work is the establishment of a new approach to path graph recognition (i.e., via PR-tree construction).

The first restricted form of the PR-tree (namely, the ***strong PR-tree*** - see def. 3.2.11), characterizes the sets of directed path-tree models of graphs. Accompanying this characterization is an algorithm (see *StrongReduction* in section 5) which, for a given graph G, produces a non-empty strong PR-tree D whose directed consistent set is precisely the directed path-tree models of G (i.e., $\overrightarrow{\mathcal{T}}_G$) iff G is a directed path graph. Additionally, this algorithm runs in $\mathcal{O}(a(n+m) * n * m)$ time and the strong PR-trees that it produces is linear in size with respect to the input graph. Moreover, the only difference between *Reduction* and *StrongReduction* is that the latter requires a slightly more restrictive post-processing step. This algorithm also provides a new approach to directed path graph recognition.

The ***rooted PR-tree*** is the other restricted form of the PR-tree and it characterizes the sets of path-path models of a given graph respectively (see theorem 5.3). We again use the PR-tree construction algorithm to produce a rooted PR-tree whose consistent set is precisely the path-path models of a given graph. The difference this time is that we add an extra constraint to the input. But, the rooted PR-tree is the same as the PQ-tree of Booth and Lueker [2] (see observation 3.4.2) and, consequently, is not a new characterization, but is a new proof of an old characterization.

There are many interesting open questions regarding PR-trees and related problems. In particular, we are investigating improving the efficiency and the depth of our analysis of the main algorithm presented in this work in the hopes of providing improved bounds on PR-tree construction. One such avenue is to apply the approach that Korte and Möhring have taken to simplify the construction of PQ-trees [13]. Another is the use of split decomposition. Notice that, such an improved bound would improve the best known time required to construct a PR-tree and the recognition problems of both path graphs and directed path graphs.

Furthermore, the rooted path graphs are the intersection graphs of directed paths in rooted trees and they also have a clique tree theorem [18]. This graph class is contained strictly between the interval and path graphs and is another candidate for characterization via PR-trees.

8 Acknowledgements

The author would like to thank his thesis supervisor, Dr. Derek Corneil, and the Ontario Graduate Scholarship program and the Natural Science and Engineering Research council of Canada for financial assistance.

References

[1] A. AHO, J. HOPCROFT, AND J. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley. Reading, Mass, 1974.

[2] K. BOOTH AND G. LUEKER, *Testing the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms*, J. Comput and System Sci, 13 (1976), pp. 335–379.

[3] S. CHAPLICK, *Characterizing and recognizing path graphs and directed path graphs using pr-trees (addendum)*. `http://cs.toronto.edu/~chaplick/PRtreesAddendum.pdf`, 2011.

[4] S. CHAPLICK, M. GUTIERREZ, B. LÈVÊQUE, AND S. TONDATO, *From path graphs to directed path graphs*, in 36th Workshop on Graph-Theoretic Concepts in Computer Science (WG'10), Lecture Notes in Computer Science 6410 (2010) 256-265, 2010.

[5] D. G. CORNEIL, S. OLARIU, AND L. STEWART, *The lbfs structure and recognition of interval graphs*, SIAM Journal on Discrete Mathematics, 23 (2009), pp. 1905–1953.

[6] E. DAHLHAUS, *Private communications*, 2007.

[7] E. DAHLHAUS AND G. BAILEY, *Recognition of path graphs in linear time*, in The Fifth Italian Conference on Theoretical Computer Science, (Revello, 1995), World Sci. Publishing,River Edge, NJ, 1996, pp. 201–210.

[8] D. FULKERSON AND O. GROSS, *Incidence matrices and interval graphs*, Pacific Journal of Mathematics, 15 (1965), pp. 835–855.

[9] F. GAVRIL, *The intersection graphs of subtrees of trees are exactly the chordal graphs*, Journal of Combinatorial Theory Series B, 16 (1974), pp. 47–56.

[10] ——, *A recognition algorithm for the intersection of graphs of paths in trees*, Discrete Mathematics, 23 (1978), pp. 211–227.

[11] P. GILMORE AND A. HOFFMAN, *A characterization of comparability graphs and interval graphs*, Canad. J. Math., 16 (1964), pp. 539–548.

[12] E. KÖHLER, *Graphs without Asteroidal Triples*, PhD thesis, Dept. of Mathematics – Technical University of Berlin., Berlin, Germany, 1999.

[13] N. KORTE AND R. H. MÖHRING, *An incremental linear-time algorithm for recognizing interval graphs*, SIAM J. Comput., 18 (1989), pp. 68–81.

[14] C. LEKKERKERKER AND J. BOLAND, *Representation of a finite graph by a set of intervals on a real line*, Fund. Math., 51 (1962), pp. 45–64.

[15] B. LÈVÊQUE, F. MAFFRAY, AND M. PREISSMANN, *Characterizing path graphs by forbidden induced subgraphs*, Journal of Graph Theory, 62, 4 (2009).

[16] G. LUEKER AND K. BOOTH, *A linear time algorithm for deciding interval graph isomorphism*, Journal of the Association for Computing Machinery, 26, No. 2 (April 1979).

[17] T. MCKEE AND F. MCMORRIS, *Intersection Graph Theory*, SIAM, 1999.

[18] C. L. MONMA AND V. K. WEI, *Intersection graphs of paths in a tree*, J. Comb. Theory, Ser. B, 41 (1986), pp. 141–181.

[19] B. S. PANDA, *The forbidden subgraph characterization of directed vertex graphs*, Discrete Mathematics, 196 (1999), pp. 239–256.

[20] D. ROSE, R. TARJAN, AND G. LUEKER, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5, No. 2: 266-283 (June 1976.).

[21] A. SCHÄFFER, *A faster algorithm to recognize undirected path graphs*, Discrete Applied Mathematics, 43 (1993), pp. 261–295.

[22] R. E. TARJAN, *Data structures and network algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983.