PQR-TREES AND UNDIRECTED PATH GRAPHS

by

Steven Chaplick

A thesis submitted in conformity with the requirements
for the degree of Masters of Science
Graduate Department of Computer Science
University of Toronto

PQR-Trees and Undirected Path Graphs

Steven Chaplick, M.Sc. 2008.
Department of Computer Science
University of Toronto

Abstract

In **[1]** Booth and Lueker introduce PQ-trees and an algorithm solving the following problem. Given a set U and a family of subsets $\{S_0, ..., S_{z-1}\}$ of U, output a data structure (PQ-tree) capturing all embeddings of U onto a **path** such that each S_i is a subpath of that **path**. Their algorithm provides a linear time recognition algorithm for interval graphs (intersection graphs of subpaths of a path) which outputs a PQ-tree capturing all *subpaths of a path* intersection representations of an interval graph.

We introduce PQR-trees, generalizing PQ-trees **[1]**, and an algorithm solving a similar problem where U is embedded in a **tree** and a **PQR-tree** is produced. Our algorithm provides a new recognition algorithm for path graphs (intersection graphs of paths in a tree) which outputs a PQR-tree capturing all *paths in a tree* intersection representations of a path graph.

Acknowledgements

I am happy to thank the people who made the writing of this thesis an enjoyable and successful process.

First and foremost I would like to thank my supervisor, Dr. Derek Corneil. His enthusiasm, support, insight, and expertise have made this work an excellent experience.

Also, I must thank my girlfriend, Marina, who has been my muse throughout this work, providing inspiration without even realising it.

Table of Contents

1 Introduction

Intersection classes of graphs have been studied by mathematicians and computer scientists for many years. In this thesis we will focus on a hierarchy of intersection classes and one intersection class in particular. Two of the most heavily studied intersection classes are the interval graphs and the chordal graphs. “Interval Graphs” can be defined to be the intersection graphs of subpaths of a path[1]. “Chordal Graphs” are defined as the graphs with no induced k-cycles with $k \geq 4$ and was not originally defined as an intersection class; however it has been shown by Gavril **[7]** that the intersection graphs of subtrees of a tree are exactly the chordal graphs. These two classes have been thoroughly studied over many years with many results, which include: a tree data structure (PQ-trees) representing an interval graph and a corresponding linear time recognition algorithm **[1]**, linear time recognition algorithm for chordal graphs **[12]**, characterizations of both with respect to their cliques **[10]**, the isomorphism completeness of chordal graphs **[9]**, linear time isomorphism for interval graphs **[9]**, and many more. Furthermore, they raise the question of “What are the intersection graphs of subpaths of a tree?” The intersection graphs of subpaths of a tree give us two more classes of graphs. The first is the “Directed Path Graphs” which are intersection graphs of directed paths in a directed tree. The second being the “Undirected Path Graphs” (we will refer to these simply as “path graphs”) which are intersection graphs of paths in a tree. The directed path graphs have been well studied, much like the interval and chordal graphs, and there have been many results published on them, some of which are: a linear time recognition algorithm **[4]**, a forbidden subgraph characterization **[11]**, their isomorphism completeness **[0]**, and a characterization with respect to cliques **[10]**. It has been shown through Gavril **[6,7]**, and Monma and Wei **[10]** that we have the following sequence of proper subsets: interval graphs $\subset$ directed path graphs $\subset$ path graphs $\subset$ chordal graphs (see **figure 1.1** for graphs that establish these strict inclusions). This is the hierarchy of intersection graphs that we will focus on and path graphs will be the specific class of interest in this thesis.

1 This is equivalent to the standard definition of interval graphs as intersection graphs of intervals of real numbers.

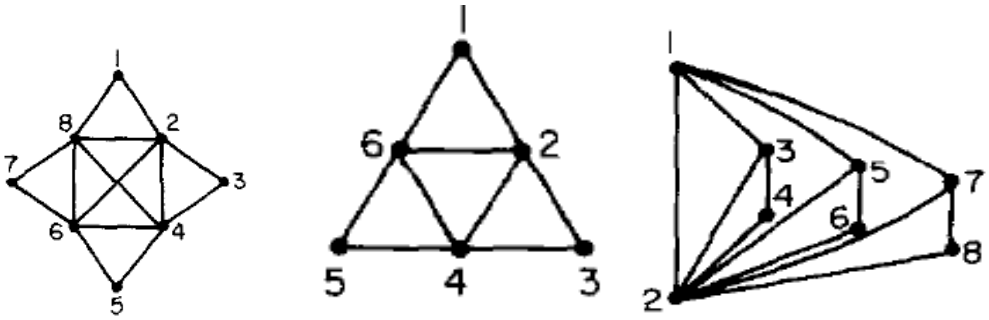


*Figure 1.1: A directed path graph (the 4-sun) which is not an interval graph (left); a path graph (the 3-sun) which is not a directed path graph (middle); and, a chordal graph (the extended claw with a universal vertex (2)) which is not a path graph (right); these three graphs were given by Monma and Wei in **[10]***

With all of the published work on these classes which are very closely related to path graphs there is relatively little that has been published on path graphs themselves. This is evident through the fact that path graphs have polynomial but not linear time recognition algorithms **[6,13]** (there is a proposed linear time algorithm by Dahlhaus and Bailey outlined in an extended abstract **[2]**, but will not appear as a journal paper **[3]**), and only a proposed forbidden characterization by Tondato et Al. **[15]** which is still incomplete **[16]**. Anecdotally, through discussions with colleagues it seems that it has not been a lack of interest in path graphs that have caused their lack of published work, but some inherent difficulty in attaining results on them.

In this thesis we present a new way to represent a path graph, via a PQR-tree, which will provide new insights into path graphs and a new approach to recognizing path graphs. This data structure is an extension to the PQ-tree developed by Booth and Lueker **[1]**.

We will first lay out the basis of definitions and notation related to path graphs in the next section. In Chapter 2 we will discuss the results from the literature regarding path graphs and related graph classes. Chapter 3 will contain the major contribution consisting of PQR-trees, a reduction algorithm that can be used to build a PQR-tree, and the relationship between PQR-trees and path graphs. In Chapter 4 we will discuss potential future work, and some structural observations on path graphs which have arisen throughout this work.

1.1 Preliminaries

By a “Graph” we mean a finite undirected graph without loops or multiple edges. For a graph G

= (V,E) we will let V = $\{v_0, ..., v_{n-1}\}$ be the vertices of G, E = $\{e_0, ..., e_{m-1}\}$ be the edges of G, and η be the number of maximal cliques in G. For $X \subseteq V(G)$, we write G[X] to be the induced subgraph of G on X. For $v \in V(G)$, we write N(v), N[v] be the open and closed neighbourhoods of v in G respectively (i.e. $N(v) = \{u \mid u,v \in V, uv \in E\}$ and $N[v] = N(v) \cup \{v\}$).

A collection of sets $\{S_0, ..., S_{n-1}\}$ is an "Intersection Representation" of a graph G=(V,E) when for all $i,j \in [0,n-1]$, $i \neq j : v_iv_j \in E$ if and only if $S_i \cap S_j \neq \emptyset$. With this notion of an intersection representation we can now formally define "the class of intersection graphs of X in Y" to be: the graphs { G | there exists an intersection representation $\{S_0, ..., S_{n-1}\}$ of G and a graph H such that $V(H) = \cup_{i=0}^{n-1} S_i$, each of $H[S_0], ..., H[S_{n-1}]$ satisfies X and H satisfies Y}. For example, the interval graphs would be the graphs { G | there exists an intersection representation $\{S_0, ..., S_{n-1}\}$ of G and a graph H such that $V(H) = \cup_{i=0}^{n-1} S_i$, each of $H[S_0], ..., H[S_{n-1}]$ is a path and H is a path}. Similarly to this definition, for graphs G and H where $\{S_0, ..., S_{n-1}\}$ is an intersection representation of G we will call $\{H, \{S_0, ..., S_{n-1}\}\}$ a/an:

- "Interval Intersection Representation of G" when each of $H[S_0], ..., H[S_{n-1}]$, and H is a path.
- "Directed Path Intersection Representation of G" when each of $H[S_0], ..., H[S_{n-1}]$ is a directed path and H is a directed tree.
- "Path Intersection Representation of G" when each of $H[S_0], ..., H[S_{n-1}]$ is a path and H is a tree.
- "Chordal Intersection Representation of G" when each of $H[S_0], ..., H[S_{n-1}]$, and H is a tree.

We can now state our motivations for extending the concept of PQ-trees with respect to path graphs. In **[1,9]** Booth and Lueker demonstrate how PQ-trees can be used for determining if a graph is an interval graph in linear time and how to decide whether two interval graphs are isomorphic in linear time. Through the recognition algorithm Booth and Lueker present for interval graphs they generate a PQ-tree, for a given graph G, that captures, up to equivalence, all the interval intersection representations of G. Furthermore, the PQ-tree also has many applications outside of interval graphs **[1]**.

The algorithm presented in **[1]** solves the following problem: given as input a set of objects U and a collection of subsets $\{S_0, ..., S_{n-1}\}$ of U, the algorithm decides whether there exists an

embedding of the elements of U onto a path such that each subset S_i occurs as a subpath of that path. The algorithm does more than just answer this question, it also provides a PQ-tree which will capture all[2] possible embeddings of U onto a path such that each subset S_i occurs as a subpath of that path.

By extending PQ-trees to path graphs we have created a data structure which, up to equivalence, captures all the path intersection representations of a path graph. The PQ-tree has provided insight into interval graphs and many other problems. For this reason we have extended it to create the PQR-tree which we expect to have a similar amount of application throughout mathematics. In particular, there are many questions with respect to path graphs that have arisen through the creation of the PQR-tree (see **Chapter 4**).

The algorithm we present (see **Chapter 3**) solves the following problem (similar to that solved in **[1]**): given as input a set of objects U and a collection of subsets $\{S_0, ..., S_{n-1}\}$ of U, the algorithm decides whether there exists an embedding of the elements of U onto a tree such that each subset S_i occurs as a path in that tree. Similarly to Booth and Lueker's algorithm, ours does more than just answer this question, it also provides a PQR-tree which will capture all[2] possible embeddings of U onto a tree such that each subset S_i occurs as a path in that tree.

2 By “all” intersection representations we mean all “clique” intersection representations as in the Clique Tree theorem[10], to be discussed in section 2.1.2.

2 Background

In this chapter we will discuss relevant results to path graphs and PQ-trees, the basis for our new data structure. We will start by discussing a variety of known results on interval graphs, directed path graphs, path graphs, and chordal graphs. Some of these include the clique tree theorem (see **theorem 2.1.2.2**), the clique separator theorem (see **theorem 2.1.2.3**), the isomorphism completeness of path graphs and chordal graphs, the best known ($O(\eta^*(n+m))$) and best proposed ($O(n+m)$) algorithms for recognizing membership in the class of path graphs, a forbidden induced subgraph characterization of directed path graphs, and a proposed forbidden induced subgraph characterization of path graphs. After that we will discuss PQ-trees as outlined in Booth and Lueker's paper [**1**].

2.1 Related / Known results

2.1.1 Chordal Graphs and Interval Graphs

Chordal graphs and interval graphs play an important role with respect to path graphs due to the following relationship complied by Monma and Wei **[10]**:

2.1.1.1 Theorem: interval graphs ⊂ directed path graphs ⊂ path graphs ⊂ chordal graphs (see figure 1.1)

We next note the following characterization of chordal graphs in terms of an intersection representation by Gavril **[7]** (this theorem is part of the proof of **theorem 2.1.1.1** above):

2.1.1.2 Theorem: [7] if G is a chordal graph then G has an intersection representation that is a collection of trees such that their union is also a tree.

With this theorem we can now relate chordal graphs to path graphs, since path graphs are the graphs with intersection representations that are collections of paths such that their union is a tree.

Asteroidal Triples are another way we can distinguish interval graphs, directed path graphs, path graphs, and chordal graphs. An "Asteroidal Triple," or AT, of a graph, $G=(V,E)$, is a triple, $A = \{a,b,c\}$, of non-adjacent vertices of G such that for each pair of vertices $\{x,y\} \subset A$, $x \neq y$, there exists an (x,y)-path, P, in G such that $P \cap N(z)$ is empty where $z \in A$ and $z \neq x$ and $z \neq y$. We call a graph G "AT-free" when G contains no asteroidal triples. Asteroidal triples provide us with the following characterization of interval graphs due to Lekkerkerker and Boland:

2.1.1.3 Theorem: [8] interval graphs are the AT-free chordal graphs.

Furthermore, in **Chapter 4** we discuss the minimal chordal ATs and some more graphs which are directed path graphs but not interval graphs, path graphs but not directed path graphs, and chordal graphs but not path graphs as in **figure 1.1**.

2.1.2 Properties of Path Graphs

Monma and Wei have produced a very detailed paper **[10]** discussing many different intersection classes of graphs. Their results on path graphs are particularly useful since they relate a path graph to its path intersection representations. In the definition of path graphs we insist that every vertex in a path graph must be a path in the graph's intersection representation and that the union of these paths must be a tree, but there is no other restrictions on how we can choose these paths in the intersection representation. For a graph G we let S_i be the set of cliques incident with v_i. In **[6]** Gavril proves the following nice property regarding the intersection representation of a path graph:

2.1.2.1 Theorem: (Clique Tree Theorem [6]) Graph G=(V,E) is a path graph if and only if there exists a tree T with vertex set C = maximal cliques of G, such that:
1. For every $i \in [0,n-1]$, $T[S_i]$ is a path; and
2. For every pair of vertices (v_j,v_k) in G, $S_j \cap S_k \neq \emptyset$ if and only if v_j is adjacent to v_k in G.

There are also corresponding clique tree theorems for interval graphs, directed path graphs, and chordal graphs given in **[10]** by Monma and Wei:

2.1.2.2 Theorem: (Clique Tree Theorem [10]) Graph G=(V,E) is an interval graph if and only if there exists a path P with vertex set C = maximal cliques of G, such that:
1. For every $i \in [0,n-1]$, $P[S_i]$ is a path; and
2. For every pair of vertices (v_j,v_k) in G, $S_j \cap S_k \neq \emptyset$ if and only if v_j is adjacent to v_k in G.

2.1.2.3 Theorem: (Clique Tree Theorem [10]) Graph G=(V,E) is a directed path graph if and only if there exists a directed tree T with vertex set C = maximal cliques of G, such that:
1. For every $i \in [0,n-1]$, $T[S_i]$ is a directed path; and
2. For every pair of vertices (v_j,v_k) in G, $S_j \cap S_k \neq \emptyset$ if and only if v_j is adjacent to v_k in G.

2.1.2.4 Theorem: (Clique Tree Theorem [10]) Graph G=(V,E) is a path graph if and only if there exists a tree T with vertex set C = maximal cliques of G, such that:
1. For every $i \in [0,n-1]$, $T[S_i]$ is a tree; and
2. For every pair of vertices (v_j,v_k) in G, $S_j \cap S_k \neq \emptyset$ if and only if v_j is adjacent to v_k in G.

We will refer to these intersection representations as clique path, clique interval, clique directed

path, and clique chordal intersection representations respectively.

Monma and Wei's contribution to path graphs involved a characterization of path graphs in terms of their clique separators, where a “Clique Separator” is a clique C in graph G=(V,E) separating G into subgraphs $G_i = (C \cup V_i)$ where s>1 and $i \in [0,s-1]$. Given a graph G with clique separator C, they call two cliques C_1 and C_2 “Antipodal with respect to C” when: they share neighbours in C, i.e. $(C_1 \cap C) \cap (C_2 \cap C) \neq \emptyset$, and when they have private neighbours in C, i.e. $(C_1 \cap C) \not\subseteq (C_2 \cap C)$, and $(C_2 \cap C) \not\subseteq (C_1 \cap C)$. Two separated subgraphs $G_1 = (C \cup V_1)$ and $G_2 = (C \cup V_2)$ are called “Antipodal with respect to C” when:

- there exists clique C_1 from G_1 and C_2 from G_2 such that $(C_1 \cap C) \cap (C_2 \cap C) \neq \emptyset$,
- there exists C_1' from G_1 and C_2' from G_2 such that $(C_2' \cap C) \not\subseteq (C_1' \cap C)$, and
- there exists C_1'' from G_1 and C_2'' from G_2 such that $(C_1' \cap C) \not\subseteq (C_2' \cap C)$.

Monma and Wei describe separating a path graph by a clique C as corresponding to dismantling its clique tree T by the the vertex C, where the subgraphs are grouped into as many categories as the degree of vertex C. Consider T as consisting of branches emanating from the vertex C; the subgraphs with subtrees on the same branch are grouped together. Two antipodal subgraphs must be in different categories. Furthermore, for any vertex v in C, the path corresponding to v extends into at most two branches. This motivates the colouring condition for path graphs given here:

2.1.2.5 Theorem: (Separator Theorem [10]) Assume clique C separates G=(V,E) into subgraphs $G_i = G[C \cup V_i]$, where s>1 and $i \in [0,s-1]$, then G is a path graph if and only if:
1. Each G_i is a path graph;
2. The G_i's can be coloured such that no antipodal pairs have the same colour; and
3. For each $v \in C$, the set of subgraphs neighbouring v is 2-coloured.

This theorem provides the basis for the current best known recognition algorithm of path graphs (see Schaffer's algorithm in **section 2.1.4**).

2.1.3 Isomorphism-completeness of Path Graphs

The problem of determining whether two graphs are isomorphic has always fascinated mathematicians and computer scientists for many reasons, two of which is its resistance to being classified as either an NP-complete or polynomial time problem and the fact that is asks the fundamental question regarding whether two constructs are equal. It has even gained its own complexity class and a problem is said to be isomorphism complete to mean that it is reducible

to and from isomorphism on arbitrary graphs. Chordal graphs are known to be isomorphism complete **[9]**. This result can also be proven through the fact that path graphs are a proper subset of chordal graphs, as in the following theorem:

2.1.3.1 Theorem: Path graphs are isomorphism complete [9].

Proof:

Consider an arbitrary connected graph G where $V(G) = \{v_0, ..., v_{n-1}\}$ and $E(G) = \{e_0, ..., e_{m-1}\}$. Let H be a graph such that: $V(H) = V(G) \cup E(G)$, and $E(H) = \{ue : u \in V(G), e \in E(G)$ such that $e = ux$ for some $x \in V(G)\} \cup \{(e, f)$: for all $e,f \in E(G)\}$. We will call H the subdivision graph of G[3], and denote it H = Subdivision(G).

Therefore, in H every subgraph consisting of a vertex, v, from G and v's edges from G will be a clique. Also, in H each edge from G will be incident with exactly three cliques (one from each of its incident vertices and the clique consisting of every edge from G).

Using this we create a path intersection representation I of H. Let $V = \{x_0, ..., x_{n-1}\} \cup \{E\}$. For each vertex, v_i, in H that is a vertex in G, we set the path of v_i to be the vertex x_i and for each vertex e in H that is an edge in G, we set the path of e to be x_a, E, x_b where $e = v_a v_b$ in G. Now every vertex of H is represented by a path in I, the union of these paths must be a tree by construction, and every edge of H is represented by two paths overlapping in I. Therefore, H is a path graph, by definition.

So, two graphs G and G' are isomorphic if and only if H and H' isomorphic where H = Subdivision(G), H' = Subdivision(G').

QED.

We next note that there exists polynomial (in fact linear) time algorithm to decide whether two interval graphs are isomorphic [9]**.** Lueker and Booth do this using PQ-trees by first creating the PQ-trees for the given graphs then comparing the PQ-trees to determine of the original graphs are isomorphic. The idea of extending their approach to superclasses of interval graphs is mentioned in their paper. However, they state that it would be optimistic to do so for path graphs given the above theorem (stating that path[4] graphs are isomorphism complete).

This isomorphism completeness result tells us that it will be optimistic to have a canonical data

3 This is not the standard notion of a Subdivision Graph

4 In their paper they discuss extending their approach to chordal graphs, but their proof of chordal graphs being isomorphism complete also applies to path graphs

structure for path graphs. However, this does not prevent us from being able to develop a data structure that can represent every clique path intersection representation of a given path graph in a tree which is polynomial in size with respect to the size of the original graph. It does mean that it would be optimistic to be able to compare two instances of this data structure in any straightforward way (i.e. most likely not polynomial time). This question is not dealt with in this thesis and is left as an open problem with respect to PQR-trees.

2.1.4 Recognizing Path Graphs

The first polynomial time recognition algorithm for path graphs was published by Gavril **[6]** in 1978. His algorithm relied primarily on the Clique Tree Theorem (see **theorem 2.1.2.2**). His algorithm ran in $O(\eta * n^3)$ time and has since been improved on to the current best known recognition algorithm by Schaffer **[13]** which is based on applying the Clique Separator Theorem (see **theorem 2.1.2.5**) in a divide an conquer approach and runs in $O(\eta * (n+m))$ time. There is also a proposed algorithm by Dahlhaus and Bailey[**2**] which is claimed to run in O(n+m) time, but has only appeared as an extended abstract. Furthermore, upon contacting Dahlhaus regarding the current progress on his proposed algorithm he stated that further work would not be done on this algorithm and it would not be appearing in a journal article **[3]**. All of these algorithms first check whether the given graph is chordal since this can be done in time O(n+m) **[14]** and simplifies the problem since these algorithms work with the maximal cliques of the given graph and the number of maximal cliques in a chordal graph is linear with respect to the size of the graph **[7]**.

We will not discuss the inner workings of these algorithms here as efficient recognition algorithms are not the focus of this thesis.

2.1.5 Forbidden Induced Subgraph Characterizations

There are two Forbidden Induced Subgraph Characterizations (FISC) that are of particular interest when discussing path graphs. First, the set of directed path graphs which is a very closely related subclass of path graphs has been characterized by a forbidden set of induced subgraphs by Panda **[11]**. Second, there is a proposed FISC of path graphs presented by Tondato et al. **[15]**.

The FISC of directed path graphs presented by Panda **[11]** is of particular interest when discussing path graphs. The following figure displays graphs A_1, ..., A_{10}, and A_{15} ($n \geq 4$) from

Panda's characterization.

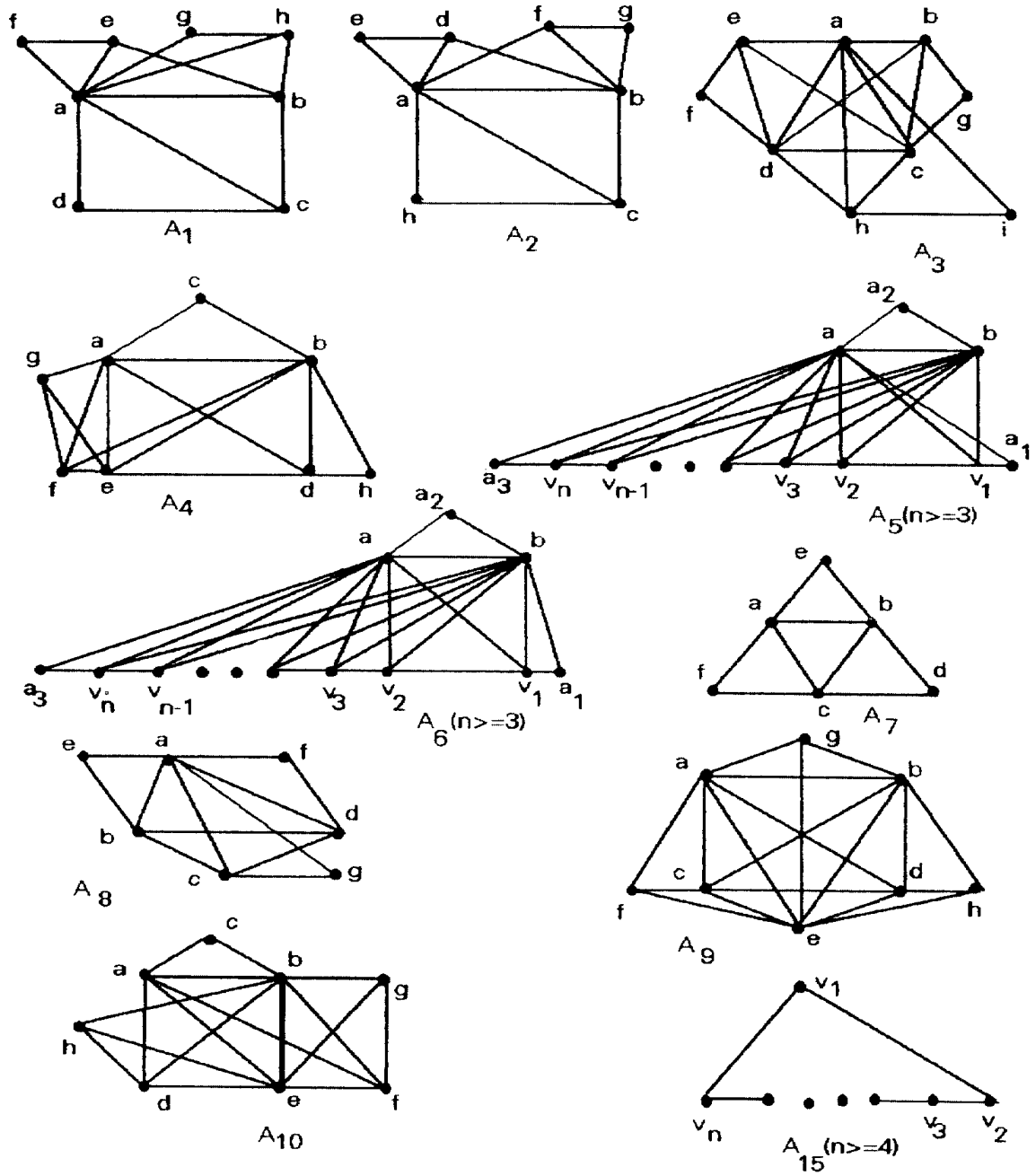


Figure 2.1: (Panda) FISC of directed path graphs (part 1) ***[11]***

In order to complete the FISC of directed path graphs we need to define some other infinite families of graphs which will appear in the list of forbidden subgraphs for directed path graphs. Panda defines A_{11} (k>1), A_{12} (k>1), A_{13} (k>1), and A_{14} (k>1) as follows:

A_{11} (k>1): $V(A_{11}(k>1)) = \{v_1, v_2, ..., v_{2k+1}, u_1, u_2, ..., u_{2k+1}, x_1\}$, k>1 such that $\{v_1, ..., v_{2k+1}\}$ is a clique, for i ∈ [2, 2k+1] u_i is adjacent to v_{i-1} and v_i, u_1 is adjacent to each of $v_1, ..., v_{2k}$, and x_1, and x_1 is adjacent to v_1.

A_{12} (k>1) (the odd sized Suns): $V(A_{12}(k>1)) = \{v_1, v_2, ..., v_{2k+1}, u_1, u_2, ..., u_{2k+1}\}$, k>1 such that $\{v_1, ..., v_{2k+1}\}$ is a clique, for i ∈ [1, 2k] u_i is only adjacent to v_i and v_{i+1}, u_{2k+1} is only adjacent to v_1 and v_{2k}, and x_1, and x_1 is adjacent to v_1.

A_{13} (k>1): $V(A_{13}(k>1)) = \{v_1, ..., v_{2k+1}, u_1, ..., u_{2k+1}\}$, k>1 such that $\{v_1, ..., v_{2k+1}\}$ is a clique, for i ∈ [2, 2k] u_i is adjacent to v_{i-1} and v_i, u_{2k+1} is adjacent to v_1 and v_{2k+1}, and u_1 is adjacent to each of $v_3, ..., v_{2k+1}$.

A_{14} (k>1): $V(A_{14}(k>1)) = \{v_1, ..., v_{2k}, u_1, ..., u_{2k+1}\}$, k>1 such that $\{v_1, ..., v_{2k}\}$ is a clique, for i ∈ [2, 2k] u_i is adjacent to v_{i-1} and v_i, u_1 is adjacent to $v_1, ..., v_{2k-1}$, and u_{2k+1} is adjacent to $v_2, ..., v_{2k-1}$.

The graphs A_{11}, A_{12}, A_{13}, and A_{14} when k=2 can be see in **figure 2.2** (below).

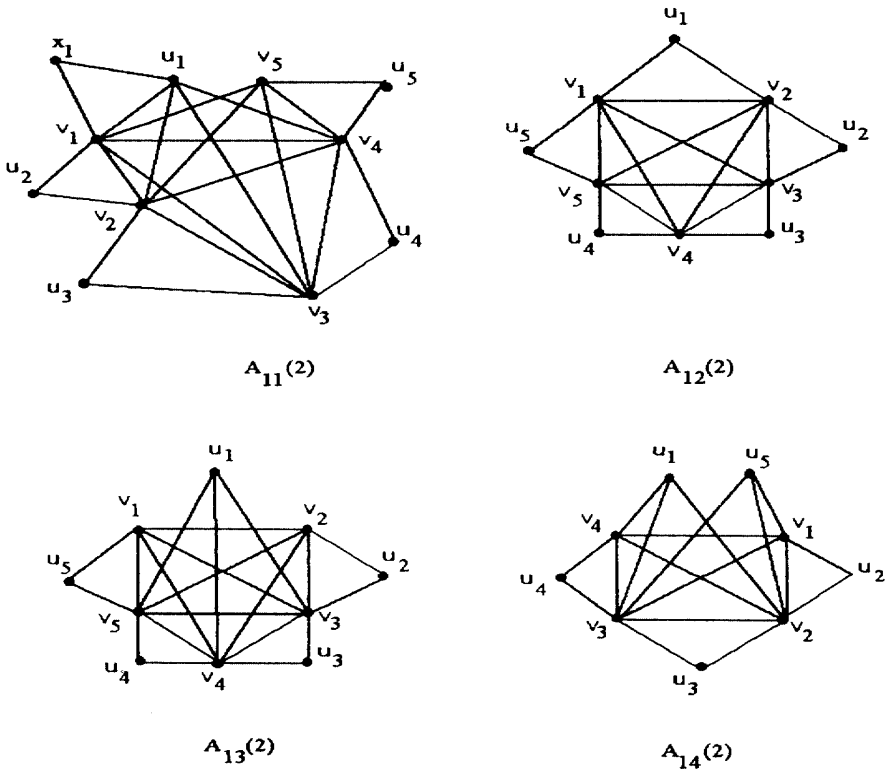


Figure 2.2: (Panda) FISC of directed path graphs (part 2) these are the graphs A_{11}, A_{12}, A_{13}, and A_{14} with k=2. ***[11]***

2.1.5.1 Theorem [11]: A graph G is a directed path graph if and only if it does not contain H $\in \{A_1, A_2, ..., A_9, A_{10}, A_{15}\}$ (as in figure 2.1) $\cup \{A_{11}(k>1), A_{12}(k>1), A_{13}(k>1), A_{14}(k>1)\}$ as an induced subgraph.

A FISC of path graphs would be quite useful for discussing their structural aspects (**Chapter 4**) and helping to prove the correctness of the new data structure we have developed (**Chapter 3**); however, the FISC characterization of path graphs is only in extended abstract form and thus not reliable enough to use as a basis for our proofs. Here it is as it appeared in **[15]**:

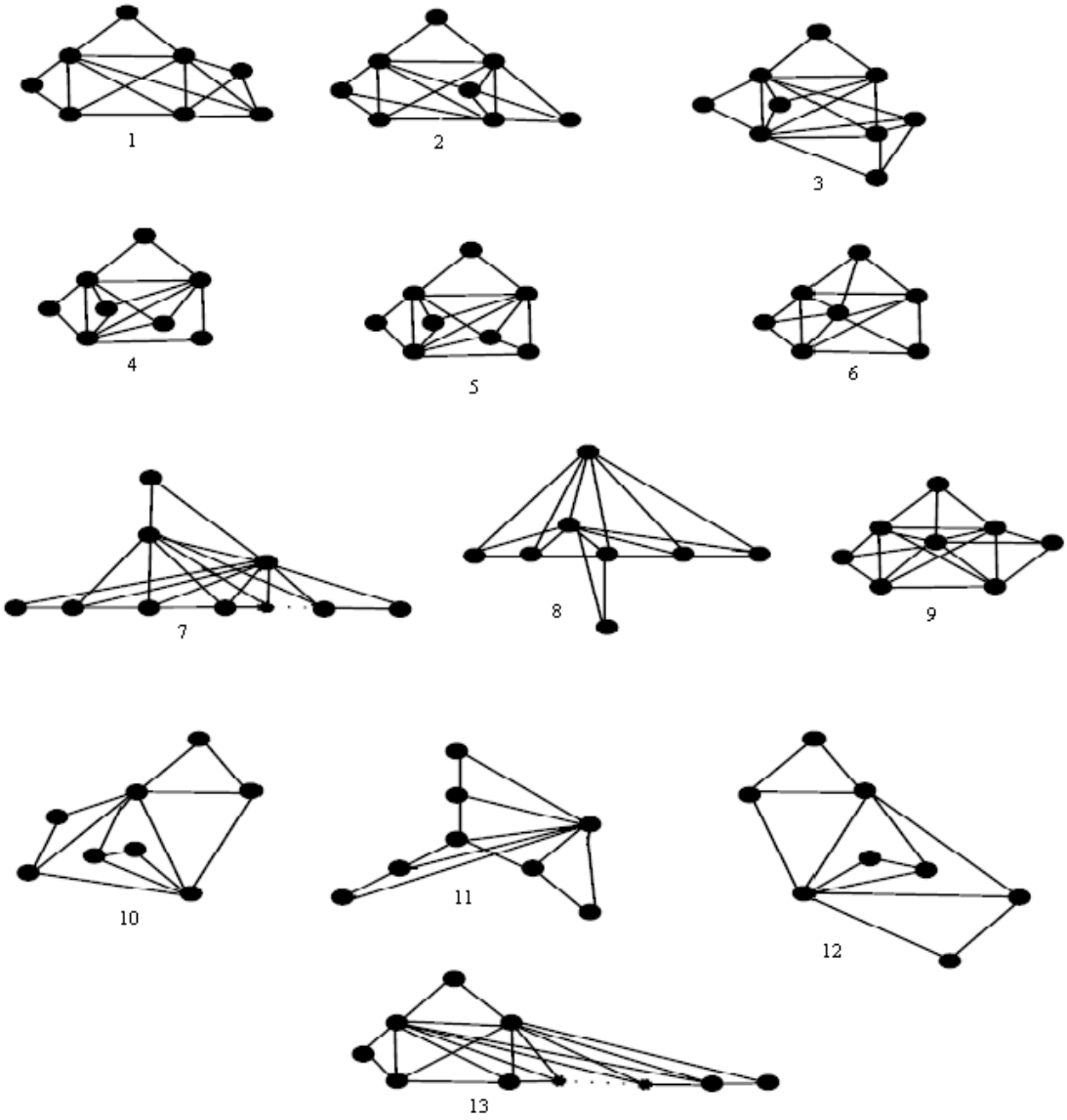


*Figure 2.3: Tondato et al. proposed FISC of path graphs **[15]** (A_{15} is purposely omitted since they are only interested in the forbidden chordal graphs with respect to path graphs).*

We note the following isomorphisms, denoted by ≈, between Tondato et al.'s set of graphs (**figure 2.3)** and Panda's (**figure 2.1**): $1 \approx A_4$, $7 \approx (\{A_8\} \cup \{A_5\})$, $8 \approx A_{10}$, $9 \approx A_9$, $10 \approx 12 \approx A_2$ (meaning that one of 10 or 12 should not be in the FISC), $11 \approx A_1$, $13 \approx A_6$. Also, graph 3 contains graph 1 as an induced subgraph, meaning that 3 should not be in the FISC. Furthermore, graphs 2, 4, 5 and 6 all contain graph A_7 as an induced subgraph (this is ok since A_7 is a path graph). We discuss this proposed characterization in more detail in **section 4.2.1**.

2.2 PQ-Trees [1]

Booth and Lueker developed a data structure called PQ-trees [**1]** which was designed with the capacity to represent the permutations of a set U in which various subsets of U must occur consecutively. More formally, a PQ-tree represents the set of permissible permutations on a set U, subject to a set of constraints, where each constraint is a subset of U which must be consecutive in all permissible permutations. If a combination of constraints does not allow any permutations, then there are no permissible permutations. Booth and Lueker showed that an iterative introduction of constraints can be used to construct a PQ-tree. This data structure is of particular interest for interval graphs because a clique interval intersection representation can be considered as U being the set of maximal cliques in a graph where each vertex v introduces the constraint consisting of the set of maximal cliques containing v. Booth and Lueker developed a set of templates (see **Appendix**) detailing modifications which must be made to a given PQ-tree in order to ensure a new constraint can be satisfied. These templates are complete, in the sense that when a constraint causes no possible permutations to be permissible it will be detected. Furthermore, they use PQ-trees and this iterative method of constructing them to determine if a given graph is an interval graph in linear time. Their algorithm also returns a PQ-tree representing the set of all possible clique interval intersection representations of the graph when it is an interval graph.

The class of PQ-trees over $U = \{a_1, ..., a_z\}$ is defined to be all rooted, ordered trees[5] whose leaves are elements of U and whose internal (non-leaf) nodes are distinguished as being either P-nodes or Q-nodes. We use the following three operations to construct a legal PQ-tree: (directly from Booth Lueker)

1. Every element $a_i \in U$ is a PQ-tree whose root is the element. The tree consists of a single leaf and is drawn as the element itself.

5 A tree where the children of every node is ordered, i.e. there is a first child, second child, third child, etc.

2. If T_1, T_2, ..., T_k are all PQ-trees then the structure shown in **figure 2.4** is a PQ-tree whose root is a P-node (drawn as a single circle).

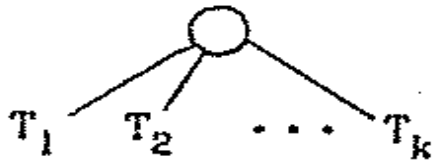


*Figure 2.4: P-node **[1]***

3. If T_1, T_2, ..., T_k are all PQ-trees then the structure shown in **figure 2.5** is a PQ-tree whose root is a Q-node (drawn as a rectangle).

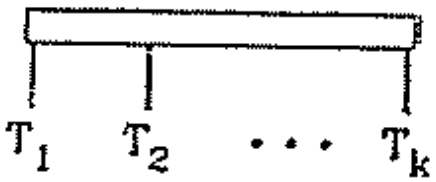


*Figure 2.5: Q-node **[1]***

A PQ-tree is considered to be "proper" if each of the following conditions are met. First, every element $a_i \in U$ appears precisely once as a leaf; this is well-defined because PQ-trees are supposed to represent permutations of a set, so it does not make sense for an element to appear more than once or to not appear at all. Second, every P-node has at least two children, this is well-defined because a PQ-tree consisting of a P-node with a single child is equivalent to simply having that node be the PQ-tree. Finally, every Q-node has at least three children, since a Q-node containing two nodes is equivalent to a P-node (this will become clear in the next paragraph) and it is useful to remove this redundancy.

So far we have not described how the P and Q nodes enforce the constraints. To do this we look at what it means for two PQ-trees, T and T', to be equivalent, denoted by $T \equiv T'$. We consider two PQ-trees equivalent if and only if one can be transformed into the other by applying zero or more "equivalence transformations", where each transformation specifies a legal reordering of the nodes within a tree. There are only two types of equivalence transformations. These are: permuting of the children of a P-node, and the reversal of the children of a Q-node. Each equivalent PQ-tree will represent a different ordering of the leaves because each PQ-tree is ordered. We will call the ordering of leaves established by the PQ-tree T the frontier of T,

denoted FRONTIER(T), and all orderings arising from equivalent PQ-trees to T will be denoted CONSISTENT(T) = {FRONTIER(T') | T' ≡ T}. In particular, there is a one-to-one correspondence between a set of permissible permutations of U and CONSISTENT(T).

From our perspective there is only one significant operation on PQ-trees. Given a PQ-tree T and subset S of the leaves of T create a new PQ-tree whose consistent permutations are the subset of CONSISTENT(T) where S is consecutive in all permutations. This operation is referred to as an S-reduction of T, and denoted REDUCE(T,S). This procedure applies a sequence of templates to the nodes of a PQ-tree. Each “template” has a “pattern” and a “replacement”. When a node matches the pattern of a template it is replaced by the template's replacement. This procedure will return a new PQ-tree. The new PQ-tree will be the null tree when the initial tree can not be reduced for the specified set.

Booth and Lueker define ROOT(T,S) for a PQ-tree T and a subset of T's leaves S, to be the least common ancestor of S in T.

This algorithm proceeds by processing the current PQ-tree, T, in a bottom-up manner. It starts by placing all of the leaves of T onto a queue. It then takes nodes off the front of the queue one at a time attempting to apply templates to them (see **Appendix**). Once every child of an internal node has been “matched” (i.e. removed from the queue), we then enqueue that internal node. If this procedure completes without returning a null tree then it will return a PQ-tree such that ROOT(T,S) will either be a P-node without any leaves that are not in S, or a Q-node with a set of consecutive children whose leaf set is exactly S.

Booth and Lueker **[1]** presented the following simplified version of this procedure:

```
Booth-Lueker-REDUCE(T,S)
|   Let Q be an empty Queue; this will contain the nodes of the PQ-tree about to be processed
|   for each X ∈ U
|   |   enqueue X on Q;
|   end for
|   while |Q| > 0
|   |   remove X from the front of Q;
|   |   mark X as matched
|   |   if some template applies to X then
|   |   |   substitute the replacement for the pattern in T;
|   |   else
|   |   |   T = null tree;
|   |   |   break
|   |   if S ⊆ {Y | X is an ancestor of Y} then
|   |   |   break;
|   |   if every sibling of X has been matched then
|   |   |   enqueue the parent of X on the Q;
|   end while
|   return T
end procedure
```

Booth and Lueker further refined this algorithm into an efficient (linear time) and more detailed version, which can be seen in their paper [**1**].

This data structure is of interest due to its application to interval graphs. The class of PQ-trees over U can be viewed as capturing all embeddings of U onto a path, P, where no two elements of U are mapped to the same node on P and the embedding is subject to a set of constraints. Each constraint is a subset of U that must be mapped to a subpath of P. Now, letting U be the set of maximal cliques from a graph, applying the Clique Tree Theorem from **section 2.1.2** and by using the sets of cliques incident with each vertex as the constraints, we can use the PQ-tree construction algorithm to determine membership in the class of interval graphs. Also, when the graph, G, is an interval graph we will get back a PQ-tree, T, such that CONSISTENT(T) contains all possible clique interval intersection representations of G.

Chapter 3 builds on PQ-trees producing a new data structure, namely PQR-trees, that captures the underlying tree structure of path graphs. The resulting data structure will form equivalence classes that can represent all possible clique path intersection representations of a given path graph.

3 PQR-trees

We now build on top of PQ-trees (discussed in s**ection 2.2)** to construct a new data structure, PQR-trees. In the same way that a PQ-tree, up to equivalence, can capture all possible clique interval representations of an interval graph, a PQR-tree, up to equivalence, will be able to capture all possible clique path intersection representations of a path graph. Much like interval graphs, path graphs can be viewed with respect to the arrangement of elements of a set, with a slight change; instead of arranging these elements in a linear fashion we are arranging them in a tree. We will still apply S-reductions similar to Booth and Lueker S-reductions to build a PQR-tree, but will require a slightly more complex algorithm.

This chapter will first define PQR-trees including equivalence transformations on them. The second section will present an algorithm describing how to construct a PQR-tree given a universal set and constraints on it. The third section presents the result that PQR-trees can be used to represent all possible path intersection representations of a path graph where the vertices in the representation correspond to the maximal cliques in the path graph.

3.1 Defining PQR-Trees

To define a PQR-tree, we first look at a definition of PQ-trees equivalent to the one presented in **section 2.2**. We start with a universal set U = $\{a_0, a_2, ..., a_{z-1}\}$. The class of PQ-trees over U can be viewed as capturing all embeddings of U onto a path, P, of length z, where no two elements of U are mapped to the same node on P and the embedding is subject to a set of constraints. Each constraint is a subset of U that must be mapped to a subpath of P. The class of well-formed PQR-trees over U is the same except instead of capturing the embeddings of the elements of U onto a path of length z, we capture the embeddings of elements of U onto a tree over z vertices (this will be proven in **section 3.2**).

We will be using the definitions of PQ-tree, P-node, Q-node, equivalence on PQ-trees, ROOT(T,S), FRONTIER(T), and CONSISTENT(T) introduced by Booth and Lueker and presented in **section 2.2**. For a P-node P = $\{p_0, ..., p_{k-1}\}$, $p_0, ..., p_{k-1}$ will be the children of P ordered 0 to k-1. For a Q-node Q = $\{q_0, q_1, ..., q_{k-1}\}$, $q_0, ..., q_{k-1}$ will be the children of Q ordered 0 to k-1.

We need to introduce some new definitions and notation on PQ-trees before being able to

formally define the PQR-tree. We first note that given a PQ-tree T, with FRONTIER(T) = $t_0 t_1 ... t_{k-1}$, its frontier can also be represented as an undirected path whose vertices are the leaves of T (i.e. $t_0, t_1, ..., t_{k-1}$) and t_i is adjacent to t_{i+1} for all i in [0, k-2], we will call this path PATH(T). Given a Q-node, $Q=\{q_0, q_1, ..., q_{k-1}\}$, we will call the subsequence $S=\{q_i, q_{i+1}, ..., q_j\}$ a "sub-Q-node of Q" where $0 \leq i \leq j \leq k-1$. An extension of this definition is "proper-sub-Q-node", which is a sub-Q-node, S, of Q-node, Q, such that there is at least one child of Q that is not a child of S. For a PQ-tree T, a subset, S, of its leaves, and X = ROOT(T,S)~~w~~ we define "T restricted to S", denoted $T|_S$ as either X when X is a P-node, or the minimal sub-Q-node of X such that its leaves contain S when X is a Q-node. Finally, for a Q-node $Q=\{q_0, ..., q_{k-1}\}$ we will call the "internal children of Q," denoted I(Q), the children $\{q_1, ..., q_{k-2}\}$ (Note: when k = 2, $I(Q) = \emptyset$)[6].

An ordered collection of PQ-trees, $T = \{T_0, ..., T_{k-1}\}$ with leaf sets $L_0, ... L_{k-1}$ respectively, is said to be a "loop" when for all $i \in [0, k-1]$, $L_i \cap L_{(i+1) \bmod k} \neq \emptyset$. We will now introduce the "Helly Property on a collection of PQ-trees". A collection of PQ-trees, $T = \{T_0, ..., T_{\gamma-1}\}$, satisfies the Helly property if for all ordered subsets $T' = \{T'_0, ..., T'_{k-1}\}$ of P, if T' is a loop then~~:~~

1. there is a common subset of leaves, L, shared by $T'_0, ..., T'_{k-1}$; and
2. for all $i,j \in [0,k-1]$ $T'_i|L \equiv T'_j|L$.

~~there is a common subset of leaves shared by $T'_0, ..., T'_{k-1}$.~~ In **figure 3.1** below we see two collections of PQ-trees, one which does not satisfy the helly property and one which does.

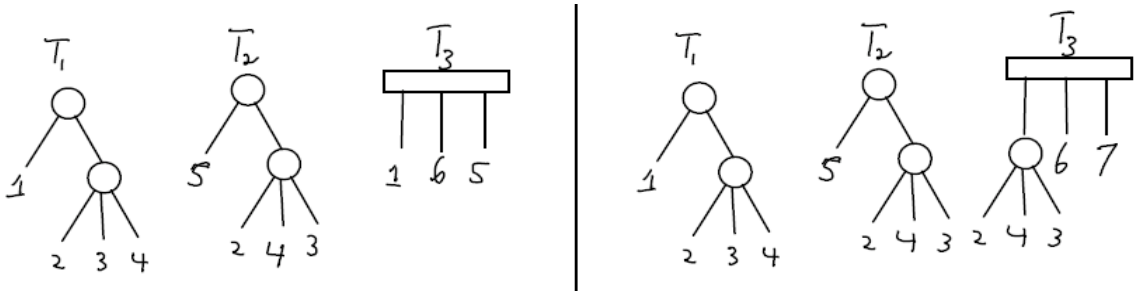


Figure 3.1: The PQ-tree collection (left) {T_1, T_2, T_3} does not satisfy the Helly Property because T_1, T_2, and T_3 form a loop but they do not have a common set of leaves, and the PQ-tree collection (right) {T_1, T_2, T_3} does satisfy the Helly Property because T_1, T_2, and T_3 form a loop and they have a common subset of leaves {2,3,4}.

3.1.1 Definition[7]: A **PQR-tree T** with leaves $L = \{c_0, ..., c_{\eta-1}\}$ is a collection of PQ-trees

6 We will allow Q-nodes to have 2 children, this means that the PQ-trees will not necessarily be proper.

7 ~~Note: t~~The context of the "***R***" in PQR-tree will become clear later when the well-formedness of PQR-trees is

$\{T_0, ..., T_{\gamma-1}\}$ with leaf sets $\{L_0, ..., L_{\gamma-1}\}$ respectively, where T satisfies the following conditions:

1. The leaves of T are exactly the union of the leaves of T's PQ-trees; formally, $L = \cup_{i=0}^{\gamma-1} L_i$.
2. When two PQ-trees in T share a common subset of leaves the PQ-tree rooted at the least common ancestor of those leaves must be a subtree of both trees; formally, for all $i, j \in [0, \gamma-1]$, if $A = L_i \cap L_j \neq \emptyset$ then $T_i|_A$ is equivalent to $T_j|_A$.[8]
3. The collection $\{T_0, ..., T_{\gamma-1}\}$ satisfies the Helly Property.

A PQR-tree T is said to be "well-formed" when every pair of PQ-trees that share a common subset of leaves enforce the same ordering on that common set of leaves; formally, for all $i, j \in [0, \gamma-1]$, if $A \subseteq (L_i \cap L_j)$ and $A \neq \emptyset$ then $PATH(T_i|_A)$ enforces the same ordering on A as $PATH(T_j|_A)$. We will refer to this property as "TWF". **Figure 3.2** below shows one PQR-tree which is TWF and one which is not.

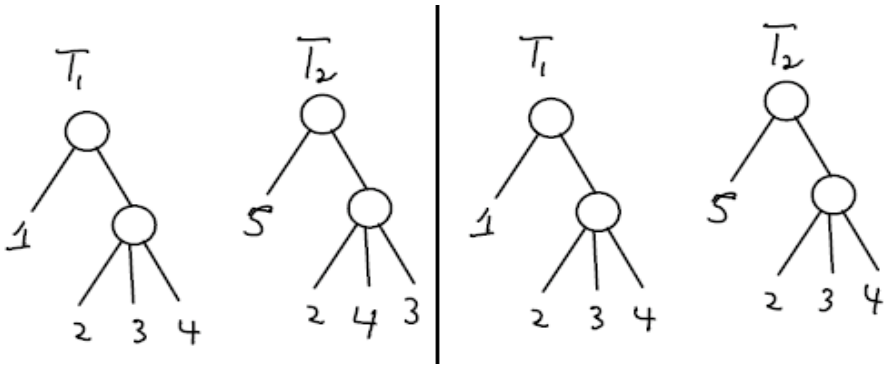


Figure 3.2: The PQR-tree (left) $\{T_1, T_2\}$ is not TWF because the shared leaves 2,3,4 do not occur in the same order in both of T_1 and T_2, and the PQR-tree (right) $\{T_1, T_2\}$ is TWF because the shared leaves 2,3,4 occur in the same order in both of T_1 and T_2 and these are the only shared leaves.

With PQR-trees defined, we can now extend the concept of frontier from PQ-trees to a collection of PQ-trees and thus to PQR-trees. The frontier of T, a collection of PQ-trees, denoted FRONTIER(T), will be the graph that is the union of all the paths corresponding to the PQ-trees; formally, for a collection T of PQ-trees $T_0, ..., T_{\gamma-1}$, $FRONTIER(T) = \cup_{i=0}^{\gamma-1} PATH(T_i)$. Since a PQR-tree T is just a constrained collection of PQ-trees we will use this definition for PQR-trees as well as for arbitrary collections of PQ-trees. We note the following lemma regarding frontiers of collections of PQ-trees:

discussed.

8 This is just a special case of the Helly property (i.e. when loops have size 2), but we list it separately because this specific case is quite important in our proofs.

3.1.2 Lemma: Given a collection of PQ-trees T and a collection of nodes X = {X_0, ..., X_{k-1}} from the PQ-trees in T, FRONTIER(X) will be a subgraph of FRONTIER(T).

Proof:

This follows from the definition of FRONTIER.

QED

We now discuss what it means for two PQR-trees to be equivalent, denoted T ≡ T'. We consider two PQR-trees equivalent if and only if one can be transformed into the other by applying zero or more "equivalence transformations". The Equivalence transformations on PQR-trees are exactly the same as those on PQ-trees (see **section 2.2**). In particular, any equivalence transformation on a PQR-tree T = {T_0, ...,$T_{\gamma-1}$} would be an equivalence transformation on a PQ-tree T_i for some i ∈ [0, γ-1]. Note that these transformations do not guarantee TWF will be maintained.

To extend the concept of the "consistent set of a PQ-tree" (defined in **section 2.2**) to PQR-trees we cannot simply take all PQR-trees that are equivalent to a given PQR-tree T because this set may contain PQR-trees whose frontiers are not trees and we are only trying to capture the possible embeddings of the leaves of T onto a tree subject to the path constraints. As such we will define the "consistent set of a PQR-tree T", denoted CONSISTENT(T), to be all PQR-trees that are equivalent to T and whose frontiers are acyclic (we do not insist that they are trees since it is possible to have a forest). Now, we must determine which PQR-trees can have equivalent PQR-trees satisfying this condition. We will prove that for a PQR-tree T, CONSISTENT(T) = {T' | T' ≡ T and T' is TWF}.

3.1.3 Theorem: Given PQR-trees T, T', if T' ≡ T then T'∈ CONSISTENT(T) if and only if T' is TWF

To prove this we will prove the following equivalent Theorem:

3.1.4 Theorem: (TWF) Given a PQR-tree T, T is TWF if and only if FRONTIER(T) does not contain a cycle.

Proof:

We will prove this by first showing that when T is not TWF FRONTIER(T) contains a cycle, then showing when T is TWF FRONTIER(T) does not contain a cycle.

To prove the first part we assume T is not TWF. Since T is not TWF it must contain two PQ-trees T_1 and T_2 with leaf sets L_1 and L_2 respectively, such that $A = L_1 \cap L_2$, A and P_1 = PATH($T_1|_A$) is different from P_2 = PATH($T_2|_A$). Since P_1 and P_2 have the same vertex set, A =

$\{a_0, ..., a_{k-1}\}$, there must be an edge $e = a_i a_j$ in P_1 that is not in P_2 where $i \neq j$, $i, j \in [0,k-1]$, therefore FRONTIER($\{ ROOT(T_1, A), ROOT(T_2, A)\}$) contains a cycle. Thus, FRONTIER(T) must contain a cycle by **lemma 3.1.2**.

Now we prove the second part. To do this we assume T is TWF and show that FRONTIER(T) must not contain a cycle. We will do this by inductively combining the PATHs of the PQ-trees in T. Let $T = \{T_0, ..., T_{\gamma-1}\}$ and G_i = FRONTIER($\{T_0, ..., T_{i-1}\}$). So we will prove "G_h has no cycles" for all $h \in [1, \gamma]$ by induction on h.

We start by examining the base case G_1 is just PATH(T_0) and thus has no cycle. We assume that G_i has no cycle for all $0 < i < \gamma$. Now, we need to prove G_{i+1} has no cycles. So, $G_{i+1} = G_i \cup PATH(T_{i+1})$ by construction. We know that G_i has no cycles by our assumption. Suppose for a contradiction that G_{i+1} contains a cycle. So, PATH(T_{i+1}) contains a subpath $X= x_0 x_1 ... x_{k-1}$, for some $k>1$, that is edge disjoint from a component C of G_i, and shares exactly vertices x_0 and x_{k-1} with C and in C there is a path $Y = y_0 ... y_{w-1}$, where $y_0 = x_0$ and $y_{w-1} = x_{k-1}$ for some $w>1$ (note: when $2 = k = w$, $X = Y = x_0 x_1$, and $X \cup Y = X$; thus X would not introduce a cycle). Therefore, one of these paths must contain internal vertices disjoint from the other. Without loss of generality let $D = \{T_0, ..., T_{z-1}\}$ be the elements from $\{T_0, ..., T_{i-1}\}$ where $d \in D$ implies PATH(d) $\cap$ Y contains at least one edge. Therefore, $D' = (D \cup T_{i+1})$ is a loop and by the Helly property and the fact that T is TWF there must be a common subset, $Z = \{z_0, ..., z_{a-1}\}$, among the leaf sets of the elements in D' and PATH(Z) must be a subpath of the paths for every element in D'. Furthermore, without loss of generality assume K is the x_0 to z_0 subpath of PATH(T_{i+1}) and K includes x_{k-1}. Now, consider $T_j \in D$, that contains x_0, and let L be the subpath of PATH(T_j) from x_0 to z_0. But now PATH(T_{i+1}) and PATH(T_j) must enforce the same ordering $(L_{i+1} \cap L_j) \supseteq \{x_0, z_0\}$ since T is TWF, but K from PATH(T_{i+1}) is different from L from PATH(T_j), therefore T cannot be TWF giving us a contradiction. Thus, G_{i+1} has no cycles and by induction FRONTIER(T) = G_γ has no cycles, giving us the result that when T is TWF FRONTIER(T) has no cycles.

Therefore, T is TWF if and only if FRONTIER(T) has no cycles.

QED.

The next interesting question regarding the consistent set of a PQR-tree T is what conditions on

T cause CONSISTENT(T) to be empty. To examine this question we will look at a particular substructure within PQR-trees. We first need to define an operation on Q-nodes; we define the "intersection of two Q-nodes Q_1 and Q_2 from a PQR-tree" (denoted $Q_1 \cap Q_2$) to be the largest[9] sub-Q-node[10] that has equivalent sub-Q-nodes in both Q_1 and Q_2. An "R-collection" is an unordered collection of Q-nodes, $\{q_0, ..., q_{k-1}\}$, from a PQR-tree such that for all i,j $\in$ [0,k-1] and $i \neq j$ q_i is not a descendant of q_j and q_i is not a sub-Q-node of q_j. An R-collection, R, is called "well-formed"[11] when it has **no** ordered, odd sized subset $X=\{x_0, ..., x_{z-1}\}$ where $L_0, ..., L_{z-1}$ are the leaf sets of x_0 to x_{z-1} respectively, such that:

1. $z > 2$;
2. the sub-Q-node $w = \cap_{i=0}^{z-1} x_i$ with leaf set L_w has $|L_w| > 1$;
3. for all $i \in [0, z-1]$ w is a proper-sub-Q-node of $(x_i \cap x_{(i+1) \bmod z})$, i.e. L_w is a proper subset of $(L_i \cap L_{(i+1) \bmod z})$; and
4. for all $i \in [0, z-1]$ $w = (x_{(i-1) \bmod z} \cap x_i \cap x_{(i+1) \bmod z})$, i.e. $L_w = (L_{(i-1) \bmod z} \cap L_i \cap L_{(i+1) \bmod z})$;

We will refer to the well-formed property on R-collections as RWF. **Figure 3.3** shows one R-collection that is not RWF and one that is.

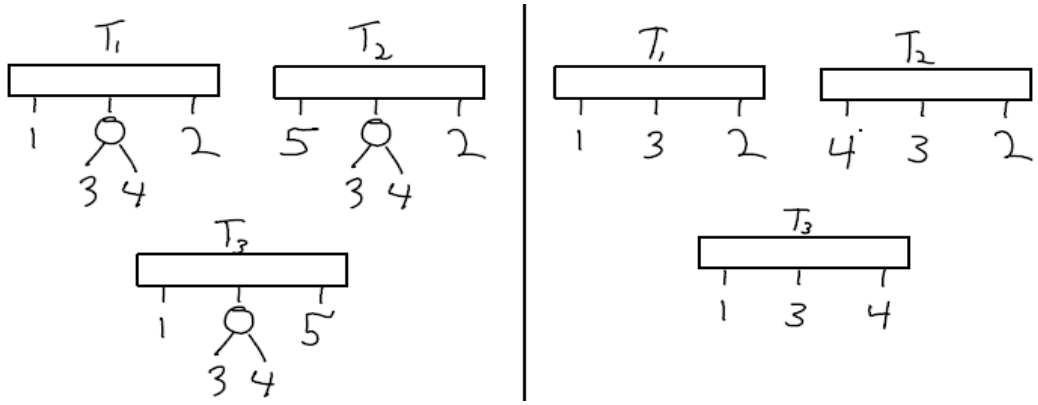


Figure 3.3: R-collection (left) $\{T_1, T_2, T_3\}$ is not RWF because $\{T_1, T_2, T_3\}$ is an odd sized proper subset and it satisfies conditions 1,2,3, and 4 from the definition, and R-collection (right) $\{T_1, T_2, T_3\}$ is RWF because the only odd sized proper subset is $\{T_1, T_2, T_3\}$ and it doesn't satisfy condition 1.

We first note the following lemma regarding R-collections in PQR-trees:

3.1.5 Lemma: Given PQR-trees T, T', if T'≡ T then T contains an R-collection, R, that is not RWF if and only if T' contains an R-collection, R', that is not RWF.

9 With respect to number of leaves
10 This may result in a sub-Q-node with only one child
11 We use "well-formedness" with respect to both R-collections and PQR-trees because these are closely related as will be shown subsequently.

Proof:

This is clear since applying equivalence transformations to an R-collection will not affect whether or not it is RWF.

QED.

Now we have a property that is maintained through equivalence transformations. So, if a PQR-tree has an R-collection that is not RWF then all of its equivalent PQR-trees will as well. Similarly, if all the R-collections within a PQR-tree are RWF, then in every equivalent PQR-tree all R-collections must be RWF. Now we prove the following:

3.1.6 Theorem: Given a PQR-tree T, CONSISTENT(T)=Ø if and only if T contains an R-collection that is not RWF.

To prove this we will prove the following Theorem (which is equivalent by **theorem 3.1.3**) that justifies the multiple use of "well-formedness":

3.1.7 Theorem: Given a PQR-tree T, there is no TWF PQR-tree T'≡ T if and only if T contains an R-collection that is not RWF.

Proof:

We will prove this by first showing that any PQR-tree T cannot be both TWF and have an R-collection that is not RWF. This will mean when T contains an R-collection that is not RWF, T cannot be TWF and no PQR-tree equivalent to T can be TWF by **lemma 3.1.5**. After that we will show when there is no TWF PQR-tree T' ≡ T, T must contain an R-collection that is not RWF.

We assume for a contradiction that T is TWF and contains an R-collection that is not RWF. Therefore, T contains an ordered R-collection R={x_0, ..., x_{z-1}} where L_0, ..., L_{z-1} are the leaf sets of x_0, ..., x_{z-1} respectively, such that: z is an odd integer greater than one, w = $\cap_{i=0}^{z-1} x_i$ and has leaf set L_w, for all i ∈ [0, z-1] w is a proper-sub-Q-node of $x_i \cap x_{(i+1) \bmod z}$, $|L_w| > 1$, and for all i ∈ [0, z-1] w = $x_{(i-1) \bmod z} \cap x_i \cap x_{(i+1) \bmod z}$. Thus $(x_{(i-1) \bmod z} \cap x_i) \cap (x_{(i+1) \bmod z} \cap x_i) = w$, so PATH($x_i$) = (..., PATH(($x_{(i-1) \bmod z} \cap x_i$) – w), PATH(w), PATH(($x_{(i-1) \bmod z} \cap x_i$) – w), ...) since T is TWF. Now without loss of generality we consider the graph G = $\cup_{i=0}^{z-2}$ PATH(x_i) (see **figure 3.4** below for the case when z=7). We can now see that adding PATH(x_{z-1}) must introduce a cycle because it must have a non-empty intersection with each of PATH(x_0) and PATH(x_{z-2}) that is outside of PATH(w), PATH(x_{z-1}) misses PATH(P_j – w) for all 0 < j < z-2, and PATH(x_{z-1}) must contain all of PATH(w) where the length of PATH(w) ≥ 2. Therefore,

the subgraph of the frontier of T corresponding to R contains a cycle, and thus the frontier of T contains a cycle. Thus, T cannot be TWF, by **theorem 3.1.4**, contradicting our assumption.

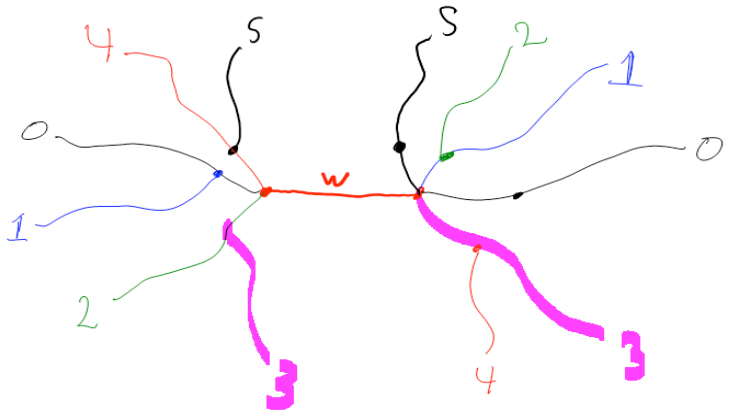


Figure 3.4: Graph G when z=7, where the line starting and ending with the same number, i, is PATH(x_i). We can see that PATH(x_6) must intersect both PATH(x_5) and PATH(x_0) on the right hand side of W and therefore must introduce a cycle.

Now, we assume there is no TWF PQR-tree equivalent to T. This means that T must not be TWF. Our approach will be to pick a set of P and Q nodes from T, which we will refer to as "sub-PQ-trees", that is "critical" and "minimal" and whose frontier contains a cycle from T, then argue that this set must be an R-collection that is not RWF. We call a set R = {Q_0, ..., Q_{k-1}}[12] of sub-PQ-trees "critical" if R is not TWF and R' = {Q_0, ..., Q_{k-2}} is TWF. Since every collection of sub-PQ-trees of size ≤ 2 from a PQR-tree has an equivalent set that is TWF (by property#2 from **definition 3.1.1**), this concept is well-defined. We call a critical set R = {Q_0, ..., Q_{k-1}} of sub-PQ-trees "minimal" if k is as small as possible and the leaf set of each $Q_i \in R$ is also as small as possible (i.e. if we can choose a proper sub-PQ-tree of Q_i instead of Q_i then we would have taken the proper sub-PQ-tree instead of Q_i). From here on we will refer to a minimal critical set of sub-PQ-trees as a minimal set.

We now take a minimal critical set of sub-PQ-trees R={Q_0, ..., Q_{k-1}} from T, where L_0, ..., L_{k-1} are the leaf sets of Q_0, ..., Q_{k-1} respectively, and let R' = {Q_0, ..., Q_{k-2}}. We will show that R is an R-collection that is not RWF. In particular, we will show that R violates at least one the four conditions on ordered subsets that cause an R-collection not to be RWF.

Notice that for all i, j $\in$ [0,k-1], $i \neq j$, Q_i will not be a sub-PQ-tree of Q_j because this would

12 At this point each Q_i may be a P-node or a Q-node, but we will show that they must all be Q-nodes.

mean k is not minimized. Similarly, if Q_i is a Q-node then Q_j will not be a sub-Q-node of Q_i because this would also mean that k is not minimized. Therefore, every overlap between a pair (Q_i, Q_j) from R will occur within either a single child of Q_i or as a sub-Q-node of Q_i and similarly for Q_j.

Next, we note that without loss of generality that R must contain a loop of size ≥ 3, since a loop is required for there to be a cycle in R's frontier. Furthermore, the largest loop in R must be R itself. This is because R is chosen to be minimal and if there were an element in R that was not involved in the largest loop, then that element would not be involved in the cycle in R's frontier. The fact that R is a loop means that there will be a common sub-PQ-tree, W with leaf set L_W, among all of the Q_i's due to the Helly property.

By examining how the PQ-trees from R overlap we see that none of them can be rooted at a P-node. The reason for this is a P-node's overlap with the other PQ-trees cannot involve more than one of its children unless the overlap contains the entire P-node itself[13] (which would contradict the minimality of R) and every overlap must contain W. Therefore, every Q_i from R which is a P-node will have a single child containing its overlaps between all other Q_j's from R, which contradicts the minimality of R with respect to leaf sets. Therefore, every element in R is a Q-node.

Thus, R must be a collection of Q-nodes that are not descendants of each other and are not sub-Q-nodes of each other; therefore, R is an R-collection where W is the sub-Q-node $\cap_{i=0}^{k-1} Q_i$. We next examine how these Q-nodes overlap. Due to the minimality of leaf sets for all $i \in [0, k-1]$, there exists $j \in [0, k-1]$, $j \neq i$ such that $L_W \subset (L_j \cap L_i)$; otherwise, Q_i would equal W and thus be contained within all other elements in R contradicting the minimality of R.

Claim#1: For all $i \in [0, k-1]$ there exists $x,y \in [0, k-1]$ $x \neq y$, $x \neq i$, $y \neq i$, such that $L_W \subset A = (L_x \cap L_i)$, $L_W \subset B = (L_y \cap L_i)$ and $A \not\subseteq B$ and $B \not\subseteq A$ and $(A \cup B) = L_i$.

Suppose Claim#1 is false for some $i \in [0, k-1]$. This means there exists $x \in [0, k-1]$ $x \neq i$, such that $L_W \subset A = (L_x \cap L_i)$, and for all $y \in [0, k-1]$ $x \neq y$, $y \neq i$, $B = (L_y \cap L_i)$, and $B \subseteq A$. But, now all of the intersections between Q_i and all other elements in R are contained in Q_x;

13 this is due to property#2 from definition **3.1.1**

therefore, Q_i is contained within Q_x by the minimality of leaf sets, and this contradicts the minimality of |R|. Therefore, Claim#1 must be satisfied by R.

Claim#2: For all $i \in [0, k-1]$, there exists $x,y \in [0, k-1]$ $x \neq y$, $x \neq i$, $y \neq i$, such that $L_W \subset (L_x \cap L_i)$, $L_W \subset (L_y \cap L_i)$, $(Q_x \cap Q_y \cap Q_i) = W$, and $L_i \subset L_x \cup L_y$.

We first note that there must be x and y such that $L_i \subset L_x \cup L_y$ due to the fact that without loss of generality some Q_x must share Q_i's "left-most leaf" and some Q_y must share Q_i's "right-most leaf". Suppose Claim#2 is false for some $i \in [0, k-1]$. From Claim#1 we know there exist $x,y \in [0, k-1]$ $x \neq y$, $x \neq i$, $y \neq i$, such that $L_W \subset A = (L_x \cap L_i)$, $L_W \subset B = (L_y \cap L_i)$ and $A \not\subseteq B$ and $B \not\subseteq A$ and $(A \cup B) = L_i$. Without loss of generality we can write Q_i, Q_x, and Q_y as the following sequences of sub-Q-nodes: $Q_i = \{a, c_1, W, c_2, b\}$, $Q_x = \{..., a, c_1, W, c_2, ...\}$, and $Q_y = \{..., c_1, W, c_2, b, ...\}$, where a, W, b cannot be empty and at most one of c_1 and c_2 may be empty. Now, we know that $R'' = R - \{Q_i\}$ must be TWF by the minimality of R, but by property#2 from **definition 3.1.1** the sub-Q-node $\{a, c_1, w, c_2\}$ from Q_i must be equivalent to the same sub-Q-node in Q_x and similarly for Q_y. Therefore, we can apply equivalence transformations to Q_i such that $PATH(Q_i)$ is a subpath of $PATH(Q_x) \cup PATH(Q_y)$, but this means that there is a set of equivalence transformations we can apply to R to make R TWF giving us a contradiction. Therefore, Claim#2 must be satisfied by R.

Claim#3: $|L_W| > 1$. (note: we already know $|L_W| > 0$ by the Helly property since R is a loop).

Suppose $|L_W| = 1$. By Claim#2, for Q_{k-1} there exists $x,y \in [0, k-2]$ $x \neq y$,, such that $L_W \subset (L_x \cap L_{k-1})$, $L_W \subset (L_y \cap L_{k-1})$ and $(Q_x \cap Q_y \cap Q_i) = W$. Now, without loss of generality we write Q_{k-1}, Q_x, and Q_y as the following sequences of sub-Q-nodes: $Q_{k-1} = \{a, W, b\}$, $Q_x = \{..., a, W, ...\}$, $Q_y = \{..., W, b, ...\}$. Since R' is TWF, therefore $PATH(Q_x) \cup PATH(Q_y)$ is acyclic. Furthermore, we can apply equivalence transformations to the sub-Q-nodes a and b of Q_{k-1} such that their paths in Q_{k-1} are the same as their paths in Q_x and Q_y (this is by property#2 from **definition 3.1.1**). Now $PATH(Q_{k-1}) = PATH(\{a, W\}) \cup PATH(\{W, b\})$ since the overlap between $PATH(\{a, W\})$ and $PATH(\{W, b\})$ is a single vertex. In **figure 3.5** below we can see that it is possible to add $PATH(Q_{k-1})$ to G' without introducing a cycle, therefore $|L_W| > 1$. (note: when $|L_W| > 1$ performing this union of paths may introduce a cycle as will be seen in Claim#4)

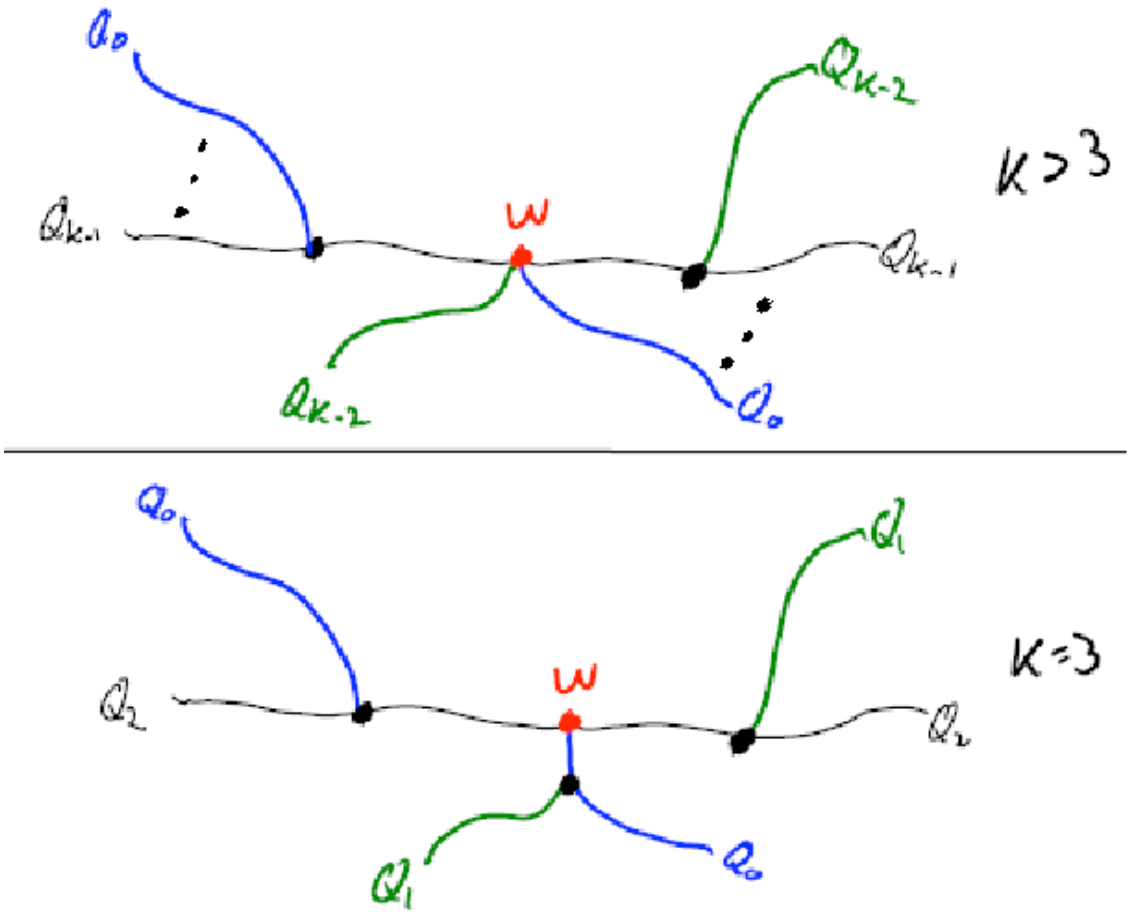


Figure 3.5: PATH(Q_{k-1}) when $|L_W| = 1$, when k=3 (bottom) Q_0 and $Q_1 = Q_{k-2}$ overlap outside of W, but when k>3 (top) Q_0 and Q_{k-2} will not overlap outside of W.

Claim#4: For all i ∈ [0, k-1] there are sub-Q-nodes a_i and b_i of Q_i such that a_i and b_i have exactly one child and $Q_i = \{a_i, W, b_i\}$.

Suppose Claim#4 is false for some i ∈ [0, k-1]. So, without loss of generality let $Q_i = \{a'_i, a_i, W, b_i, b'_i\}$ where one of a'_i and b'_i has at least one child, and a_i and b_i have exactly one child. Consider R" = R – $\{Q_i\}$, we know R" is TWF by the minimality of R and is therefore acyclic. Now, we consider the cycle $C = \{c_1, ..., c_v\}$ in FRONTIER(R" ∪ Q_i). Since for all i ∈ [0, k-1] PATH(W) is a proper subpath of PATH(Q_i), and (PATH(Q_i) ∩ C) ≠ ∅ , therefore by the minimality of R, PATH(W) is a proper subpath of C. For i, consider x,y satisfying Claim#2; therefore without loss of generality $Q_x = \{..., a'_i, a_i, W, ...\}$ and $Q_y = \{..., W, b_i, b'_i, ...\}$. Furthermore, without loss of generality PATH($\{a'_i, a_i, W\}$) = (d_1 ... d_α w_1 ... w_β) and PATH({W,

$b_i, b'_i\}) = w_\beta \ldots w_1\, e_1 \ldots e_\delta$ where $PATH(W) = w_1 \ldots w_\beta$ (Note: by Claim#3 $\beta > 1$) and by property#2 from definition **3.1.1** we can apply equivalence transformations to Q_i such that $PATH(Q_i) = PATH(\{a'_i, a_i\}) \cup PATH(\{W, b_i, b'_i\}) \cup \{\text{the edge } d_\alpha w_\beta\}$ (Note: $PATH(Q_i)$ must contain the edge $d_\alpha w_\beta$ otherwise it would not introduce a cycle) (see **figure 3.6**). But, this means that $C = \{d_\alpha, w_1, \ldots, w_\beta\}$ and will always consist of one leaf from a sub-Q-node "adjacent" to W and all of the leaves from W. Therefore, by the minimality of leaf sets in R neither a'_i and b'_i can have any children and thus Claim#4 must be satisfied.

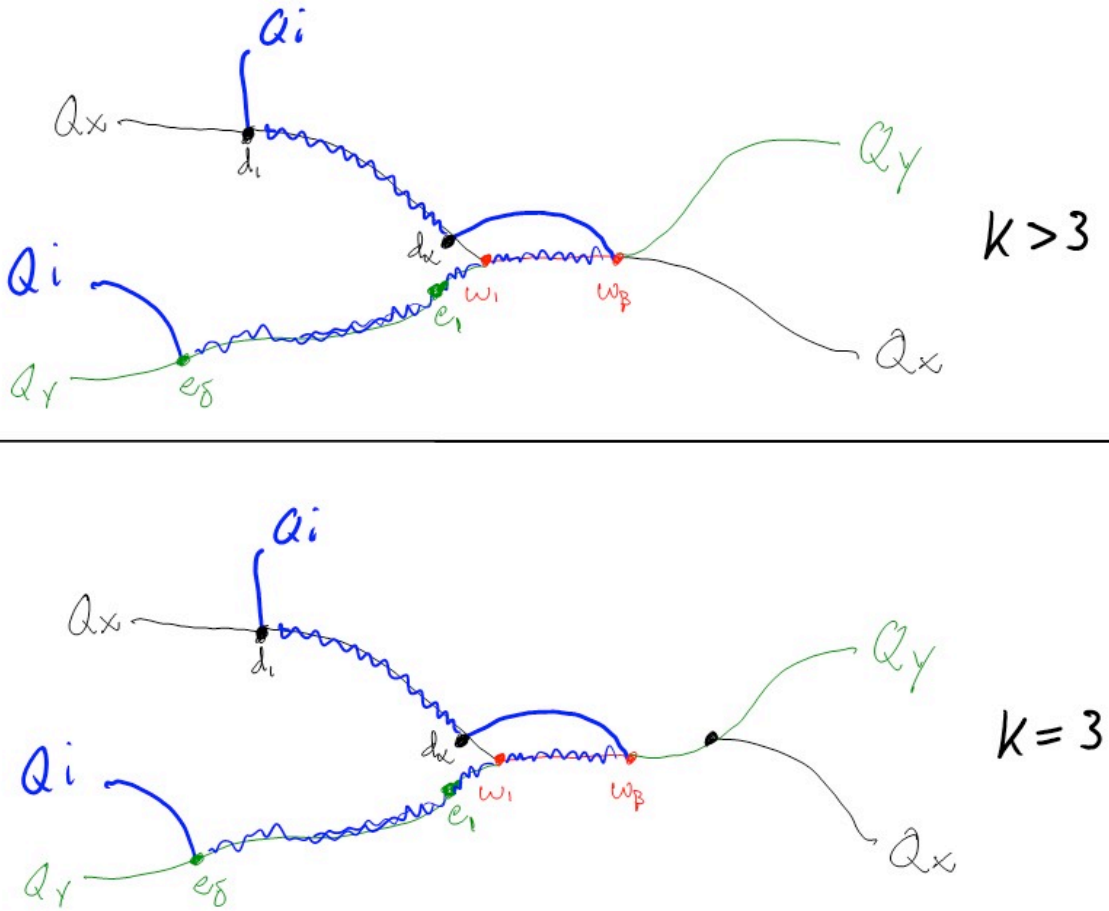


*Figure 3.6: Creating $PATH(Q_i)$ using $PATH(Q_y)$ and $PATH(Q_x)$ (similarly to **figure 3.5** the difference here is when k=3 (bottom) Q_x and Q_y will overlap outside of W and when k>3 (top) Q_x and Q_y will not overlap outside of W.*

This leads to the following:

Corollary#1: Given Q_i, Q_x, and Q_y satisfying Claim#2, and P_x and P_y from FRONTIER(R –

$\{Q_i\}$) where $P_x = PATH(Q_i \cap Q_x)$ and $P_y = PATH(Q_i \cap Q_y)$, if w_1 has degree = 2 in P_x then w_1 has degree = 2 in P_y and w_β has degree = 1 in both P_x and P_y. In particular, w_β is an endpoint of both P_x and P_y (and similarly when w_β has degree = 2).

We now need to order the elements of R such that for all $i \in [0, k-1]$ $L_W \subset (L_i \cap L_{(i+1) \bmod z})$, $W = (Q_{(i-1) \bmod k} \cap Q_i \cap Q_{(i+1) \bmod k})$. We will call this property Π. Iteratively applying Claim#2 we can create a subset of the elements of R satisfying Π, as follows:

1. Initialize $S = \{Q'_0, Q'_1, Q'_2\}$ where $Q'_1 = Q_1$ and Q'_0 and Q'_2 are the corresponding Q_x and Q_y as in Claim#2 with respect to Q_1. Let i = 2.
2. Apply Claim#2 to Q'_i to get Q'_{i+1} different from Q'_{i-1}[14].
3. if $Q'_{i+1} \notin S$
   - add Q'_{i+1} to S
   - i = i+1
   - Go to 2.
4. Otherwise $Q'_{i+1} = Q'_j$ for some $j \in [0, i-2]$ and $S' = \{Q'_j, ..., Q'_i\}$ is an ordered subset of R that satisfies Π.

Therefore, by the finiteness of R, R must contain at least one subset S' satisfying Π.

If |S'| is odd, then S' is an R-collection that is not RWF and therefore S' is not TWF (from the "<=" direction of this proof, above). Therefore, if |S'| is odd then by the minimality of R, S' = R. Furthermore, an odd sized S' cannot contain a smaller set satisfying Π, since this would contradict the minimality of R (i.e. without loss of generality this set would be odd since this would be exactly like adding a chord to an odd sized cycle). Therefore, when |S'| is odd we have an R-collection that is not RWF.

(*) Suppose R contains no odd sized subset satisfying Π. By the construction above, R must contain a subset S' satisfying Π. Therefore, R must contain an even sized subset satisfying Π.

An "(a,b)-sequence in R" for $a,b \in [0,z-1]$ $a \neq b$ is a subset of R that is a sequence $S = \{Q'_0, ..., Q'_{\lambda-1}\}$ such that $Q_a = Q'_0$ and $Q_b = Q'_{\lambda-1}$, $Q_j \in R$ implies Q_j occurs at most once in S, and for all $i \in [1,\lambda-2]$ W is a proper sub-Q-node of $(Q'_{(i-1)} \cap Q'_i)$ and $(Q'_i \cap Q'_{(i+1)})$, and $(Q'_{(i-1)} \cap Q'_i \cap Q'_{(i+1)}) = W$.

14 Claim#4 tells us that without loss of generality if X, Y and X', Y' both satisfy Claim#2 with respect to Q_j then X, Y' satisfy Claim#2 with respect to Q_j and therefore, by Claims #2 and #4: Given Q'_i with $Q_x = Q'_{i-1}$ there exists Q_y such that Q_x and Q_y satisfy Claim#2 with respect to Q'_i.

Let $S^R_i = \{Q_j \mid$ there exists an (i,j)-sequence in R starting from Q_i and ending at $Q_j\} \cup \{Q_i\}$

Consider R' = R – $\{Q_{k-1}\}$. As we saw in **figure 3.6** from Claim#4 in order for PATH(Q_{k-1}) = PATH($\{a_{k-1}, W, b_{k-1}\}$) = ($[a_{k-1}]_1$ ... $[a_{k-1}]_c$ w_1 ... w_β $[b_{k-1}]_1$ ... $[b_{k-1}]_d$), where PATH(W) = (w_1 ... w_β), to introduce a cycle there must exist Q_x and Q_y (as in Claim#2) such that, in R', PATH(Q_x) = ($[a_{k-1}]_1$ ... $[a_{k-1}]_c$ w_1 ... w_β ...) and PATH(Q_y) = ($[b_{k-1}]_d$ ... $[b_{k-1}]_1$ w_1 ... w_β ...) (i.e. a_{k-1} and b_{k-1} both occur on the "left" side of W). Consider $S^{R'}_y$, now we have two cases:

Case#1: $Q_x \notin S^{R'}_y$: for all $Q_i \in S^{R'}_y$, PATH(Q_i) = ($[a_i]_1$ ... $[a_i]_{\alpha(i)}$ w_1 ... w_β $[b_i]_1$ ... $[b_i]_{\delta(i)}$), apply the equivalence transformation reversing w_1 ... w_β. This will not create a cycle in frontier R' because we are just reversing W in every path. But now PATH(Q_y) = (... w_1 ... w_β $[b_{k-1}]_1$ ... $[b_{k-1}]_d$) and applying this transformation to all Q_y's[15] will mean that PATH(Q_{k-1}) is a subgraph of FRONTIER(R') and thus FRONTIER(R) is acyclic. This contradicts the fact that R must not be TWF by **theorem 3.1.4**. Therefore, Case#1 cannot happen and for i = k-1 there must be x,y satisfying Claim#2 such that $Q_x \in S^{R'}_y$.

Case#2: $Q_x \in S^{R'}_y$: Now for all (y,x)-sequences S from R', |S| must be odd (otherwise (S ∪ $\{Q_{k-1}\}$) would be an odd sized subset of R satisfying Π, which would contradict (*)).

Let S = $\{(Q_y =) Q'_0, Q'_1, ..., Q'_{\lambda-1} (= Q_x)\}$, $Q_i = Q'_{-1} = Q'_\lambda$, and

for all i ∈ [0, λ], let $X_i = \begin{Bmatrix} \text{Left}, \textit{when } w_\beta \text{ is an endpoint of PATH}(Q'_i \cap Q'_{i-1}) \\ \text{Right}, \textit{otherwise} \end{Bmatrix}$ [16](i.e. X_i = "Left" when $Q'_i \cap Q'_{i-1}$ occurs to the "left" of W and similarly for X_i = "Right").

Notice that when i ∈ [1, λ-2] and $X_i = X_{i+1}$ there must be a cycle in FRONTIER(R') since Q'_{i-1} and Q'_{i+1} intersect Q'_i on the same "side" of W (**see figure 3.6**). Since λ is odd, (λ-1) must be even; therefore when $X_1 = X_{\lambda-1}$ there must exist i ∈ [1, λ-2] such that $X_i = X_{i+1}$, but this implies FRONTIER(R') contains a cycle which contradicts the fact that R' must be TWF by **theorem 3.1.4**. Thus, (**) for all i ∈ [1, λ-2] $X_i \neq X_{i+1}$ and $X_1 \neq X_{\lambda-1}$.

Suppose without loss of generality $X_0 = X_\lambda$ = "Left" = X_1 (Note: they must be equal since Q_{k-1} must create a cycle in FRONTIER(R)). Therefore, $X_{\lambda-1}$ = "Right" and ($Q'_1 \cap Q'_0$) = ($Q'_0 \cap Q_{k-1}$) by (**) and Claim#4. Thus, Q'_1 and Q_x satisfy Claim#2 with respect to Q_i and S'' = $\{Q_{k-1}$,

15 Every Q_y such that for every Q_x, $Q_x \notin S^{R'}_y$, where Q_x and Q_y satisfy Claim#2 with respect to Q_{k-1}.
16 As seen in Corollary#1

$Q'_1, ..., Q'_{\lambda-1}\}$ satisfies Π. However, λ is odd, therefore |S"| is odd and now S" is an odd subset of R satisfying Π which contradicts (*). Therefore, Case#2 cannot happen.

Therefore, (*) cannot happen, meaning R must contain an odd sized subset satisfying Π; thus, R must be an R-collection that is not RWF.

Therefore, when T has no TWF equivalent PQR-tree T', T must contain an R-collection that is not RWF.

QED.

We have now shown that the consistent set of a PQR-tree T will contain all PQR-trees equivalent to T whose frontiers are trees, and we have shown that this set will only be empty when T contains a malformed R-collection. Furthermore, we know that a PQR-tree is TWF if and only if all R-collections in T are RWF. Therefore, we can refer to a PQR-tree as being well-formed without ambiguity regarding what this really means.

3.2 PQR-tree Construction

In this section we describe an algorithm to construct a PQR-tree T over a set $U = \{u_1, u_2, ..., u_z\}$ subject to constraints $S_i \subseteq U$ for $i \in [1,n]$ such that the consistent set of T contains all possible trees whose vertices are U and in every tree every S_i appears as a path. The approach we will take to constructing such a PQR-tree will be to enforce each constraint one at a time in an iterative manner, where the first iteration will be applied to the PQR-tree $T = T(U,\emptyset)$ which is the PQR-tree that contains one PQ-tree, T_i, for each element a_i in U and T_i is just the leaf u_i.

For a PQR-tree T we let T_i denote its i^{th} PQ-tree and let L_i be the leaves of T_i. We will also let γ be the number of PQ-trees in T.

Each iteration can be broken down into two major stages when enforcing a constraint S on T, the current tree. The first step is to ensure that in each PQ-tree T_i, from T, the leaves of T_i that are in the set S will appear consecutively in every frontier in CONSISTENT(T_i). This can be achieved by applying the reduction algorithm created by Booth and Lueker to T_i (**see section 2.2**). So, the first step will be to apply Booth and Lueker's reduction algorithm to every PQ-tree in T (it is important to note that when such a reduction on one of the PQ-trees is not possible, the algorithm will stop and an empty PQR-tree will be produced). The second stage of the algorithm will be to ensure the entire set S appears consecutively in every frontier in CONSISTENT(T).

Since we have applied Booth and Lueker's reduction algorithm to each PQ-tree from T we will now have at most one node X_i from each PQ-tree, T_i, such that X_i is T_i restricted to S. We now consider the collection containing all such X_i's and attempt to merge these into a single PQ-tree using a templating approach similar to Booth and Lueker's reduction algorithm. Once this merging is complete we will have a PQ-tree whose leaves are exactly the set S, this PQ-tree will then be added to T. After this iteration has completed we will have a new PQR-tree T' whose consistent set will contain exactly the elements of T's consistent set in which there is a path whose vertices are exactly the vertices in S.

After the iterative part of the algorithm we will check whether the resulting PQR-tree T (assuming it is not null) is well-formed as this will tell us whether or not its consistent set is empty, by **Theorem 3.1.6**. In particular, to ensure that T is well-formed we will check for R-collections that are not RWF. We claim the consistent set of T will contain all arrangements of the elements of U as vertices of a tree where for all i ∈ [1, n] there must be a path in the tree whose vertices are exactly $S_i \subseteq U$ (this will be proved in **section 3.2.2**).

3.2.1 The Algorithm

We will use the following notation when describing the algorithm. We let T(U,S) denote the PQR-tree shown in **figure 3.7** below**.**

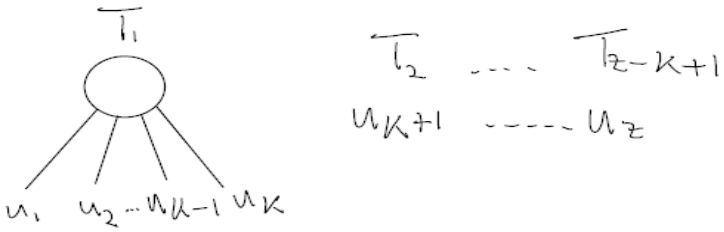


Figure 3.7: ***T(U,S): PQ-tree, T_1, which is a P-node with children = S = {u_1, ..., u_k}, and one PQ-tree for each a ∈ U-S = {u_{k+1}, ..., u_z}***

Three special cases are worth noting. T(U,∅) is the PQR-tree that contains one PQ-tree for each a ∈ U. T(U,U) is just a PQR-tree that contains one PQ-tree which has each element of U under a single P-node (Note: this is the same as T(U,U) from Booth-Lueker since it is really just a PQ-tree). T(∅,∅) is another way of specifying the null tree.

The main method (see MAIN below) of the algorithm requires a universal set U = {u_1, u_2, ..., u_z}

and a set of constraints, S_1, ..., S_n, where S_i is a constraint containing the nodes from U that must appear as a path in every tree, and will either return a PQR-tree T such that CONSISTENT(T) contains all the trees over U that satisfy S_1, ..., S_n or will return NULL when no such trees exist. This method will first determine whether the constraints can be represented by a collection of PQ-trees that will satisfying part 1 of the Helly property. Second, it will iteratively apply the constraints S_1, ..., S_n to T each time checking to make sure the reduction was successful. Finally, ~~Then~~ it will check the well-formedness of T to ensure that CONSISTENT(T) is not empty.

MAIN(U, {S_1, ..., S_n}):
| **if not** CHECK_HELLY(U, {S_1, ..., S_n}): *(described below)*
| | *Any collection of PQ-trees representing {S_1, ..., S_n} cannot satisfy the helly property*
| | return **NULL**
| **Let** T ← T(U, Ø) *(this is the initial PQR-tree for the algorithm)*
| **for each** i **in** [1, n]:
| | T = REDUCE(T,S_i) *(described below)*
| | **if** T **is NULL then**: *T has no equivalent well-formed PQR-trees satisfying S_i*
| | | **break**
| **if not** VERIFY_R_COLLECTIONS(T): *(described below)*
| | *T contains a malformed R-collection*
| | T ← **NULL**
| **return** T

To determine if the set of constraints can be represented by a collection of PQ-trees which satisfy part 1 of the Helly property we use the method CHECK_HELLY (below). In this method we build a graph G where each S_i corresponds to a single vertex and two vertices v_i, v_j are adjacent if and only if S_i intersect S_j non-empty and if G is not chordal then (U, {S_1, ..., S_n}) cannot be represented by a collection of PQ-trees satisfying the Helly property **(see Theorem 3.2.2.1)**.

CHECK_HELLY(U, {S_1, ..., S_n}):
| Let G = (V={S_1, ..., S_n}, E={(S_i S_j) : i,j ∈ [1, n], S_i ∩ S_j ≠ Ø}
| **if** G **is** chordal:
| | **return True**
| **else**:
| | **return False**

The reduction method (see REDUCE below) for PQR-trees applies a constraint S to the PQR-tree T. It will return a PQR-tree T' such that CONSISTENT(T') = CONSISTENT(T) ∩

CONSISTENT(T(U,S)) when CONSISTENT(T) ∩ CONSISTENT(T(U,S)) ≠ ∅, otherwise it will either return a malformed PQR-tree or NULL. This method will first apply Booth and Lueker's reduction method to each PQ-tree in T, then it will iteratively merge the relevant subtrees of those PQ-trees to create a new PQ-tree, R, with S as its leaves, and finally adding R to the PQR-tree T. It is important to note that as we build new PQ-trees that overlap with existing PQ-trees the overlap will be stored by reference, this way when we apply Booth and Lueker's reduction algorithm to the "shared" nodes, all operations will be applied to every PQ-tree that "contains" that node (see the **Appendix** for a more detailed discussion of this with respect to Booth and Lueker's Templates). These shared references are how we will maintain property #2 from **definition 3.1.1**. Furthermore, the Helly property will be maintained through these shared references and since new PQ-trees are only added after the application of PQ_MERGE (as discussed below).

```
REDUCE(T, S):
|   Let R be a P-node with no children (this will be the root of S in T)
|   Let L_R ← Ø (this is a list that will contain R's leaves)
|   Let C_R ← Ø (this is a list that will contain R's children)
|   Let C ← Ø (this is a list that will contain R's potential children)
|   for each i in [1, γ]:
|   |   Let S_i ← L_i ∩ S
|   |   if S_i = L_i then:
|   |   |   applying Booth-Lueker-REDUCE would not change T_i ; and
|   |   |   if REDUCE completes successfully T_i will be a subtree of R[17]; therefore,
|   |   |   add T_i to C
|   |   |   remove T_i from the list of PQ-trees in T
|   |   else if S_i ≠ Ø then:
|   |   |   Apply Booth-Lueker-REDUCE(T_i, S_i)
|   |   |   if T_i is NULL then: T_i cannot satisfy constraint S_i
|   |   |   |   return NULL
|   |   |   Let X ← T_i|_{S_i}  the full P-node / full sub-Q-node that is the root of S_i in T_i
|   |   |   if any nodes in C are descendants of X then:
|   |   |   |   remove these nodes from C
|   |   |   if X is not the descendant of a node in C then:
|   |   |   |   add X to C
|   for each X in C:
|   |   Let L_X be the leaves of X
|   |   if L_R ∩ L_X = Ø then:
|   |   |   add X to C_R
|   |   |   L_R ← L_R ∪ L_X
|   |   else:
|   |   |   Let X' ← X (X' will be the merge of X and all PQ-trees in C_R overlapping X)
|   |   |   for each P in C_R: (note: P is a PQ-tree)
|   |   |   |   Let L_P be the leaf set of P
|   |   |   |   if L_P ∩ L_X ≠Ø then:
|   |   |   |   |   remove P from C_R
|   |   |   |   |   X' ← PQ_MERGE(P, X') (described below)
|   |   |   |   |   if X' is NULL then: Merge could not combine P and X'
|   |   |   |   |   |   return NULL
|   |   |   add X' to C_R
|   |   |   L_R ← L_R ∪ L_X
|   if |C_R| = 1 then:
|   |   R ← the only node in C_R
|   else: (|C_R| > 1)
|   |   set the children of R to be C_R
|   add R to the list of PQ-trees in T
|   return T
```

17 This is the only way T_i will end up as a descendent of R and thus the only reason we would want to remove it from T's list of PQ-trees.

The input for the merge method (see PQ_MERGE below) is two overlapping PQ-trees, T_1 and T_2, where T_1 and T_2 satisfy the following: the leaves of T_1 are not contained in the leaves of T_2, the leaves of T_2 are not contained in the leaves of T_1, T_1 and T_2 share a common subset of leaves L, and $T_1|_L$ is equivalent to $T_2|_L$. It will return a single PQ-tree, X[18], satisfying the constraints of both T_1 and T_2 whose leaves are the union of the leaves of T_1 and T_2. It is important to note that the PQ-trees T_1 and T_2 may be modified during this process.

PQ_MERGE(PQ-tree T_1, PQ-tree T_2)
| **Let** L_1 be the leaves of T_1, and L_2 be the leaves of T_2
| **Let** $L \leftarrow L_1 \cap L_2$
| **Let** $R \leftarrow T_1|_L$ (*Note: R is equivalent to $T_2|_L$)*
| **Let** $X \leftarrow R$ *(X will be the result of merging T_1 and T_2)*
| **Let** L_X be the leaves of X *(Note: initially X = L)*
| **Let** P_1, P_2 be the parents of R in T_1 and T_2 respectively
| **if** P_1 = **NULL then**:
| | P_1 = ROOT(T_1, L)
| **else if** (P_1 is a Q-node) **and** (ROOT(T_1, L) = R) **and** (P_1 has a parent in T_1) **then**:
| | P_1 = the parent of P_1 in T_1
| **if** P_2 = **NULL then**:
| | P_2 = ROOT(T_2, L)
| **else if** (P_2 is a Q-node) **and** (ROOT(T_2, L) = R) **and** (P_2 has a parent in T_2) **then**:
| | P_2 = the parent of P_2 in T_2
| **while** ($L_X \neq L_1 \cup L_2$):
| | **Let** M be a "Merge Template" that applies to (R, P_1, P_2) *(see Merge Templates below)*
| | **if** there is no "Merge Template" M **then**:
| | | **return NULL**
| | (X, P_1, P_2) = M(R, P_1, P_2); Update L_X
| | **if** P_1 has a parent in T_1 **then**:
| | | P_1 = the parent of P_1 in T_1
| | **if** P_2 has a parent in T_2 **then**:
| | | P_2 = the parent of P_2 in T_2
| **return** X

The input of a Merge Template is three PQ-trees, R, P_1, P_2 where R is a sub-PQ-tree of both P_1 and P_2. The pattern in a Merge Template, M, consists of two parts and we say that "M applies to (R, P_1, P_2)" when (R, P_1) match one part of the pattern and (R, P_2) matches the other part. In particular, for a pair (R, P_i) to "match" one part of the pattern there must be a sequence of equivalence transformations that can be applied to P_i to make P_i equal to that part of the pattern. The result of applying a Merge Template is three PQ-trees, X, P_1', P_2', where CONSISTENT(P_1') $\subseteq$ CONSISTENT(P_1), CONSISTENT(P_2') $\subseteq$ CONSISTENT(P_2), R is a sub-PQ-tree of P_1' and

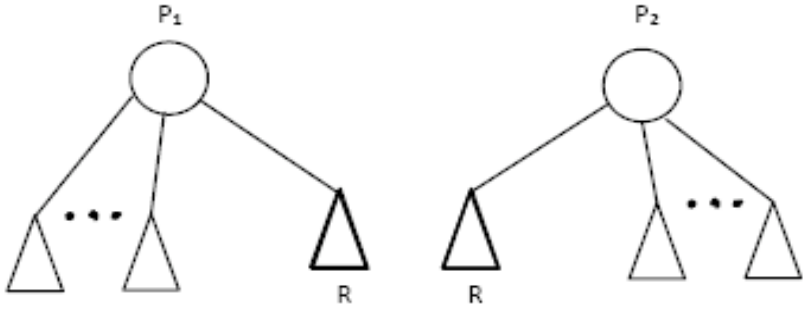
18 This method will return NULL when no such tree exists.

P_2', and P_1' and P_2' are sub-PQ-trees of X, and the leaf set of X is the union of the leaf sets of P_1 and P_2. Each Merge Template can be thought of as applying Booth-Lueker Templates (see **Appendix**) to each of P_1 and P_2 treating these nodes as though they are not the root.

As with Booth and Lueker's templates we must maintain the shared references when applying the Merge Templates (this is done in the same way as with Booth and Lueker's templates see **Appendix**).

We start by providing all the templates in which P_1 and P_2 are P-nodes, these are templates 1a, 1b: (Note: we have omitted the case when P_1 matches Merge Template #1a and P_2 matches Merge Template #1b as this is obvious given 1a and 1b).

***Pattern**: P_1 and P_2 are both P-nodes and R is not a sub-Q-node in either T_1 or T_2.*



***Replacement**: We need to apply Booth-Lueker Template P_3 (see **Appendix**) to both P_1 and P_2 in order to allow us to put them together to make X.*

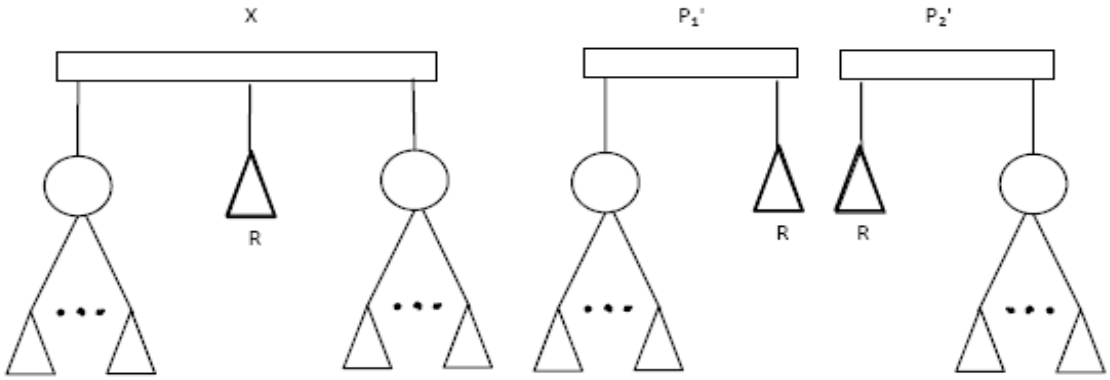
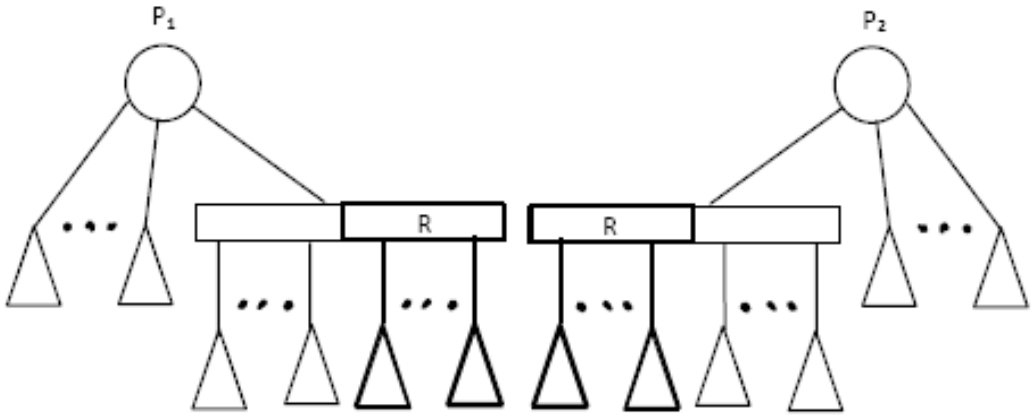


Figure 3.8: Merge Template #1a.

***Pattern**: P_1 and P_2 are both P-nodes and R is a sub-Q-node in either T_1 or T_2.*



***Replacement**: We need to apply Booth-Lueker Template P5 (see **Appendix**) to both P_1 and P_2 in order to allow us to put them together to make X.*

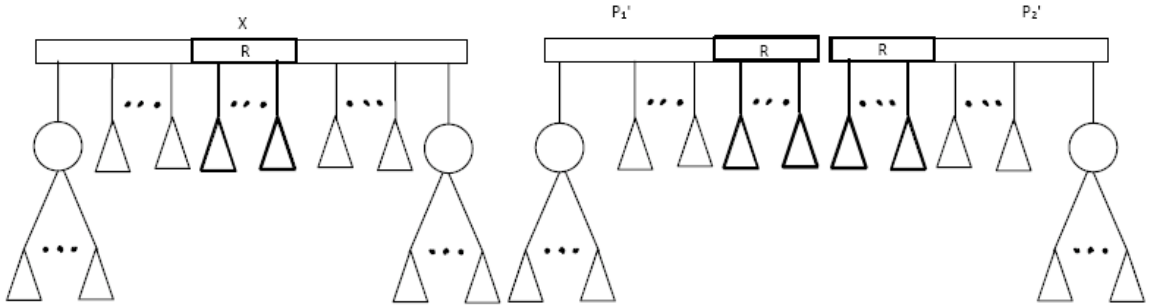
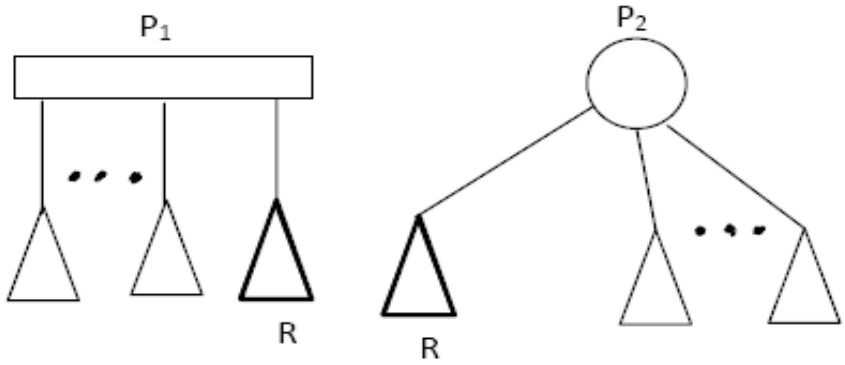


Figure 3.9: Merge Template #1b.

Next we give the templates when (without loss of generality) P_1 is a Q-node and P_2 is a P-node. These are templates 2a, 2b, and 2c: (Note: as with the first set of templates we have omitted the templates which occur as combined cases.)

***Pattern**: P_1 is a Q-node, P_2 is a P-node, and R is not a sub-Q-node.*



***Replacement**: P_1 will not be changed, but Booth-Lueker Template P3 (see **Appendix**) must be applied to P_2 in order to create X.*

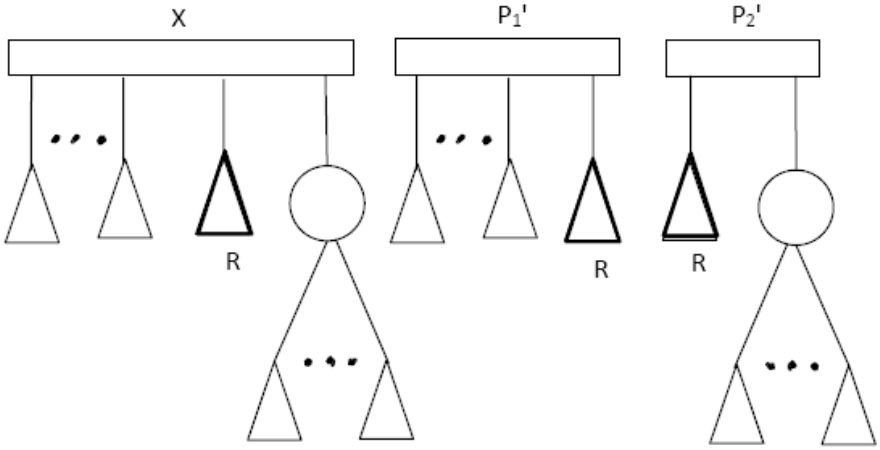
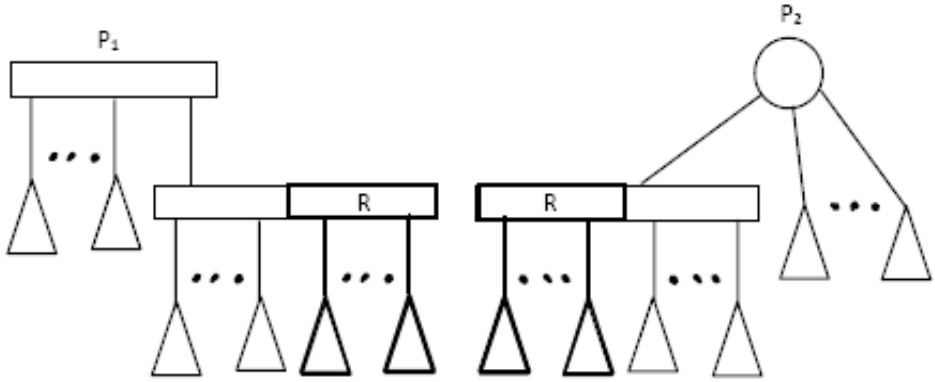


Figure 3.10: Merge Template #2a.

***Pattern**: P_1 is a Q-node, P_2 is a P-node, and R is a sub-Q-node of one child of each of P_1 and P_2.*



***Replacement**: Booth-Lueker Template Q1 (see **Appendix**) must be applied to P_1 and Booth-Lueker Template P3 (see **Appendix**) must be applied to P_2 in order to create X.*

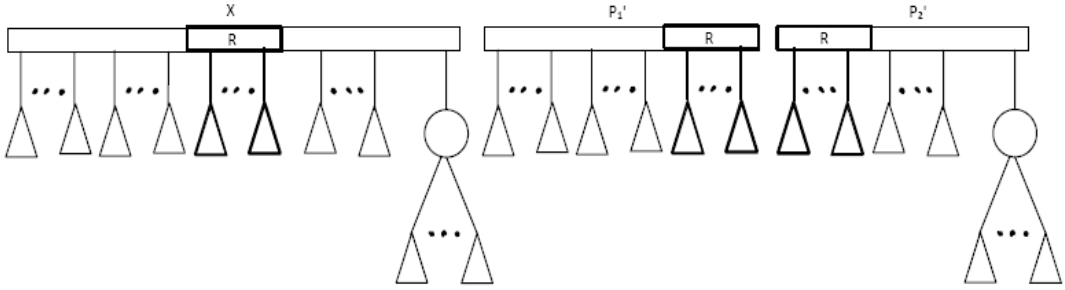
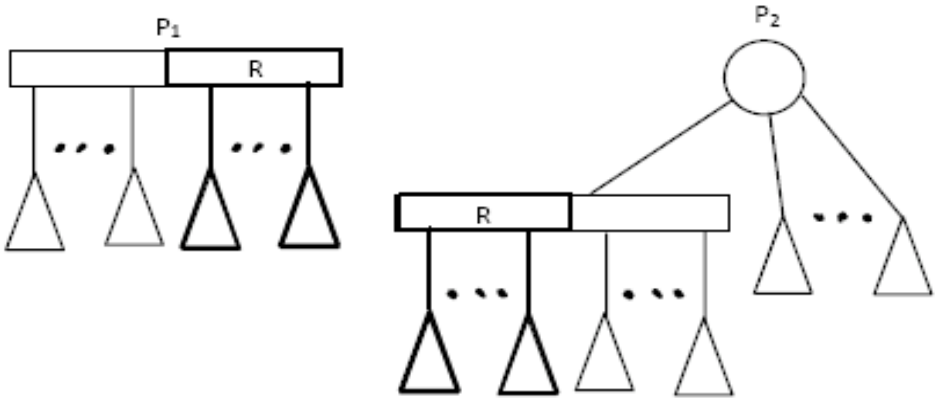


Figure 3.11: Merge Template #2b.

Pattern: P_1 is a Q-node and has no parents in T_1, P_2 is a P-node, and R is a sub-Q-node of P_1 and of a child of P_2.



***Replacement**: P_1 will not be changed, but Booth-Lueker Template P5 (see **Appendix**) must be applied to P_2 in order to create X.*

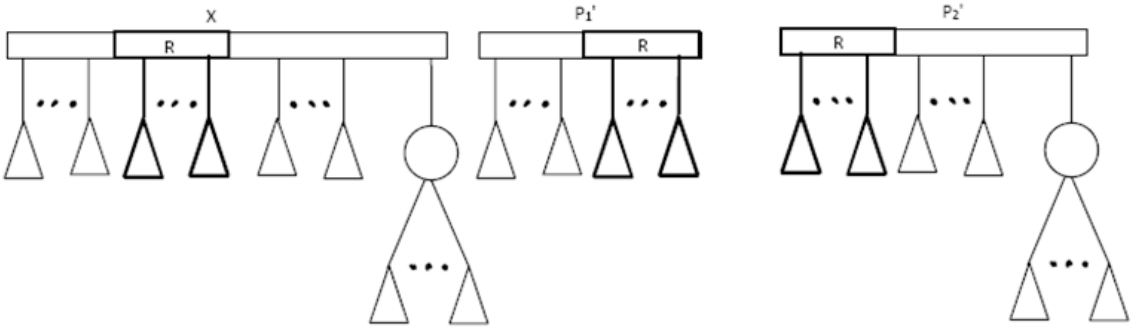
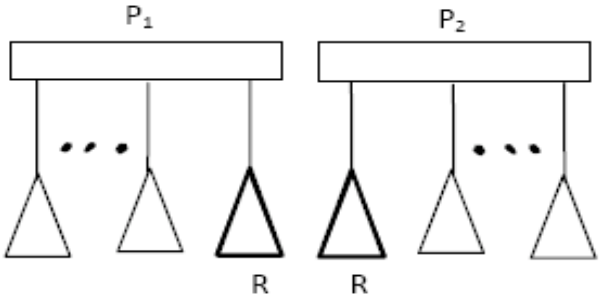


Figure 3.12: Merge Template 2c.

Finally we have the templates when both P_1 and P_2 are Q-nodes. These are templates 3a, 3b, 3c, and 3d: (Note: as with the first set of templates we have omitted the templates which occur as combined cases.)

***Pattern**: P_1 and P_2 are Q-nodes, P_1 has no parent in T_1 and P_2 has no parent in T_2, and R is a child of both P_1 and P_2.*



***Replacement**: in this case neither P_1 nor P_2 are changed (note that this will only happen when both P_1 and P_2 are the roots from their respective trees).*

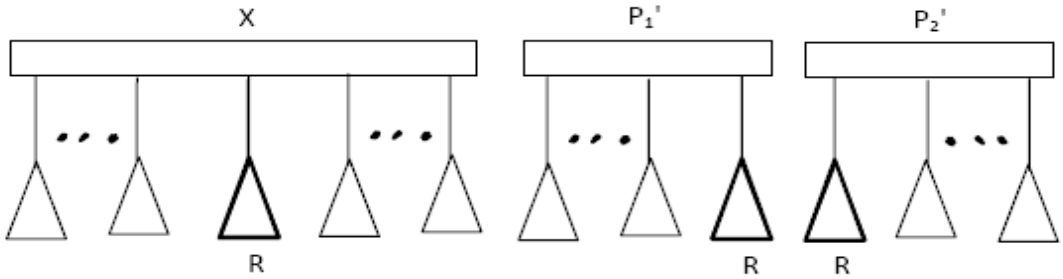
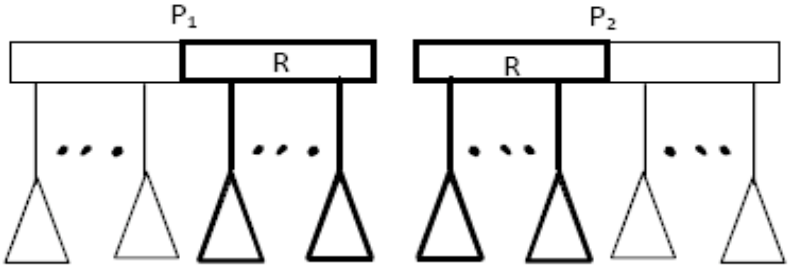


Figure 3.13: Merge Template #3a.

***Pattern**: P_1 and P_2 are Q-nodes, P_1 has no parent in T_1 and P_2 has no parent in T_2, and R is a sub-Q-node of both P_1 and P_2.*



***Replacement**: in this case neither P_1 nor P_2 are changed (note that this will only happen when both P_1 and P_2 are the roots from their respective trees).*

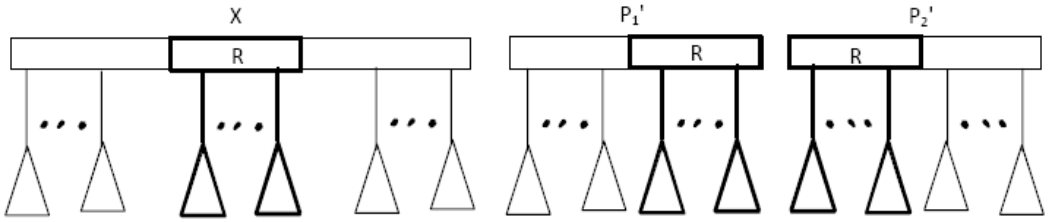
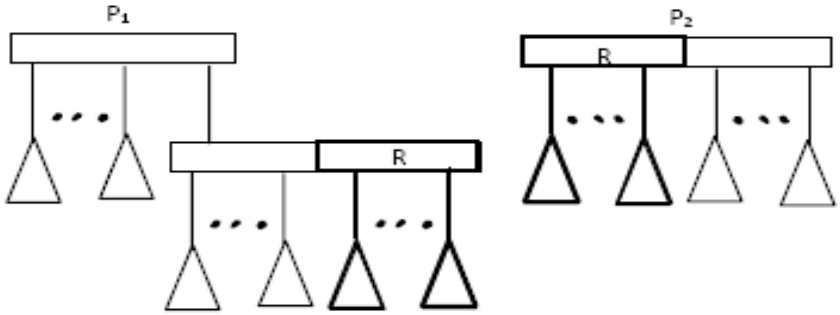


Figure 3.14: Merge Template #3b.

***Pattern**: P_1 and P_2 are Q-nodes, P_2 has no parent in T_2, and R is a sub-Q-node of P_2 and a child of P_1.*



***Replacement**: Booth-Lueker Template Q1 (see **Appendix**) must be applied to P_1, but P_2 will not be changed in order to create X.*

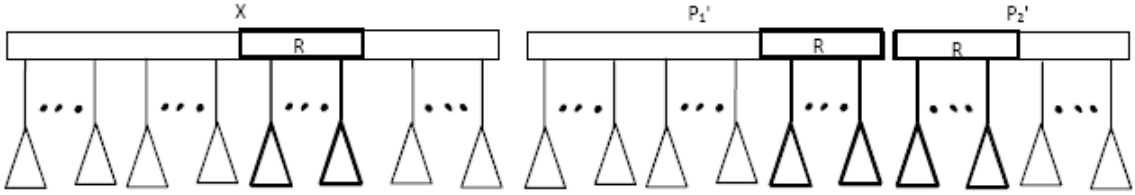
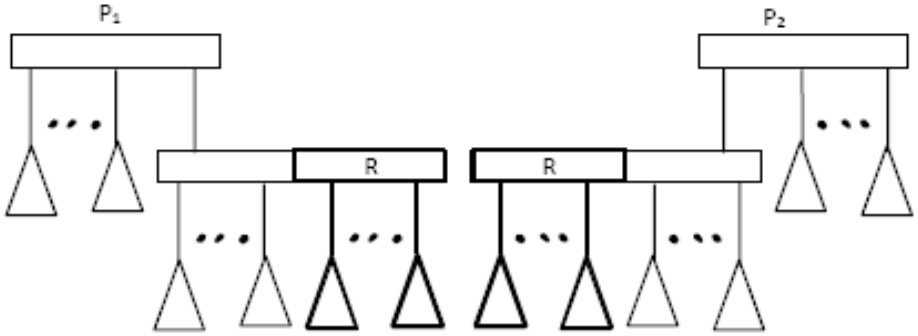


Figure 3.15: Merge Template #3c.

***Pattern**: P_1 and P_2 are Q-nodes, and R is a sub-Q-node of both a child of P_2 and a child of P_1.*



***Replacement**: We need to apply Booth-Lueker Template Q1 (see **Appendix**) to both P_1 and P_2 in order to allow us to put them together to make X.*

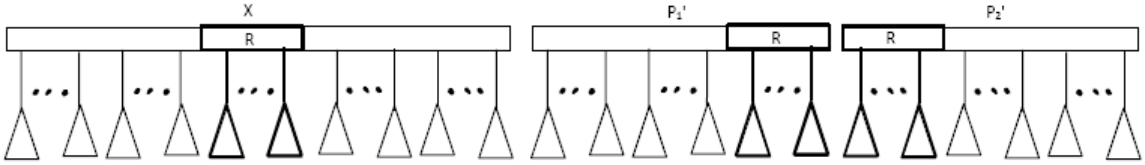


Figure 3.16: Merge Template #3d.

We have now exhausted all possibilities for the configurations of Merge Templates over P_1, P_2, and R.

Finally, we discuss the method to check the well-formedness of a PQR-tree (see VERIFY_R_COLLECTIONS below). This method takes a PQR-tree and checks each R-collection within it returning true if all of them are well-formed and false otherwise.

```
VERIFY_R_COLLECTIONS(PQR-tree T):
|   Let L be the leaves of T
|   Let Q be an empty Queue
|   Let C = ∅ (this will be the set of nodes from T processed so far)
|   for each x in L:
|   |   Let Y be the parents of x
|   |   for each y in Y:
|   |   |   if y ∉ Q then:
|   |   |   |   Enqueue y on Q
|
|   while(Q is not empty):
|   |   Let f be the front of Q
|   |   add f to C and remove f from Q.
|   |   Let Y be the parents of f
|   |   Let Z = ∅ (this will be the relevant R-collection of f)
|   |   for each y in Y:
|   |   |   if y ∉ Q and y ∉ C then:
|   |   |   |   Enqueue y on Q
|   |   |   if y is a Q-node and f ∈ I(y) then:
|   |   |   |   Let z be the smallest sub-Q-node of y such that f ∈ I(y)
|   |   |   |   add z to Z (Note: z is the Q-node {a, f, b})
|   |   V = ∅ (this will be the vertex set of an undirected graph)
|   |   E = ∅ (this will be the edge set of an undirected graph)
|   |   for each z in Z: (Note: z is the Q-node {a, f, b})
|   |   |   if a ∉ V then:
|   |   |   |   add a to V
|   |   |   if b ∉ V then:
|   |   |   |   add b to V
|   |   |   if ab ∉ E then:
|   |   |   |   add ab to E
|   |   Let G = (V,E) be an undirected graph
|   |   if G contains an Odd length induced cycle[19] then:
|   |   |   return False
|   return True
```

19 This Odd length cycle corresponds to an odd sized set satisfying property Π from theorem **3.1.7**, due to the fact that the edges from the cycle represent sub-Q-nodes from the PQR-tree that share a common child, f.

This method only checks for malformed R-collections whose overlap occurs at a sub-Q-node with exactly one child. The reason for this is because these are the only malformed R-collections that will not be detected through the application of Booth and Lueker's reduction algorithm and our PQ-Merge algorithm (see **theorem 3.2.2.32**).

3.2.2 Correctness

To prove the correctness of this approach we will first prove the correctness of CHECK_HELLY in theorem 3.2.2.1 below.~~that each iteration of MAIN will properly apply the constraint S_i to the current PQR-tree.~~

3.2.2.1 Theorem: (Chordal-Helly) Given a set $U = \{u_1, ..., u_z\}$ and a collection $\{S_1, ..., S_n\}$ of subsets of U, $G = (V=\{S_1, ..., S_n\}, E=\{(S_i\ S_j) : i,j \in [1, n], S_i \cap S_j \neq \emptyset\})$ is not chordal if and only if every collection of PQ-trees, T where FRONTIER(T) induced on S_i is a path, for all $i \in [1,n]$, does not satisfy the Helly property

Proof:

(work in progress).

Part 1: not chordal implies collection of PQ-trees not helly

Since G is not chordal, there must exist an induced k-cycle in G, k>3. Let $C = v_0, ..., v_{k-1}$ be one such cycle. Furthermore, there must be a corresponding induced cycle in the clique graph K(G).

Now we focus our attention on representing this cycle. In particular, if we can show that any collection of PQ-trees that represent this cycle cannot satisfy the Helly property then $\{S_1, ..., S_n\}$ cannot be represented by a collection of PQ-trees which satisfy the Helly property.

We first note that $G[v_0, ..., v_{k-1}]$ cannot be represented by a single PQ-tree since it is an induced cycle which is not an interval graph. Therefore, we must have at least two PQ-trees in the collection in order to represent $G[v_0, ..., v_{k-1}]$.

Since C is an induced cycle, therefore $v_i \not\subseteq v_{i+1(\text{mod } k)}$, $v_{i+1(\text{mod } k)} \not\subseteq v_i$, $v_i \cap v_{i+1(\text{mod } k)} \neq \emptyset$ and $v_i \cap v_j = \emptyset$ for all $i,j \in [0,k-1]$ where $j > i+1$ and $j < i-1$. (*)

Assume (for a contradiction) that G can be represented by a collection of PQ-trees $\{T_0, ..., T_{\gamma-1}\}$ satisfying the helly property.

Converting each PQ-tree T_i into a modified PQ-tree (as in Korte Mohring [20]), we can see that each vertex will either be represented by a P-node or as a sub-Q-node. Suppose one

vertex, v_j, is represented by a P-node in tree T_j. Now, the nodes representing $v_{j+1 \pmod k}$ and $v_{j-1 \pmod k}$ cannot belong to T_j by (*).

Part 2: collection of PQ-trees not helly implies not chordal.

Suppose G can be represented by a collection of PQ-trees, $T = \{T_0, ..., T_{z-1}\}$, that does not satisfy the helly property. We focus on a minimal collection of sub-PQ-trees, $Z = \{T'_0, ...T'_{k-1}\}$, of T that does not satisfy the helly property.

Note that for all $i,j \in [0,k-1]$, $i \neq j$, there cannot exist $P \in T$ such that T'_i and T'_j are subtrees of P since this would mean that without loss of generality $\{P, T'_{i+1}, ..., T'_{j-1}\}$ would be a smaller collection of sub-PQ-trees of T that does not satisfy the helly property.

Without loss of generality we assume the elements of Z are ordered such that for all $i \in [0, k-1]$, $T_i \cap T_{i+1 \pmod k} \neq \emptyset$. Note that for all $i,j \in [0, k-1]$, $j \notin [i-1 \pmod k, i+1 \pmod k]$, $T'_i \cap T'_j = \emptyset$. since otherwise Z would not be minimal. Consider H = FRONTIER(Z), note that H is an induced subgraph of FRONTIER(T).

QED.

Next, we prove that each iteration of MAIN will properly apply the constraint S_i to the current PQR-tree.

3.2.2.2 Theorem: (PQR-tree) Let T be a PQR-tree over a set U and let S be any subset of U. Then CONSISTENT(REDUCE(T,S)) = CONSISTENT(T)∩ CONSISTENT(T(U,S))

Proof: *(mostly direct from Booth and Lueker [1] differences in bold italics)*

To show these two sets are the same it is sufficient to prove that each is contained within the other.

The first step is to demonstrate that the left-hand side is contained within the right-hand side. This has two parts, we must show that REDUCE(T,S) is equivalent to a tree in CONSISTENT(T) and to a tree in CONSISTENT(T(U,S)).

Let π be a ***graph*** in the class CONSISTENT(REDUCE(T,S)). If such a ***graph*** exists the tree must have been successfully reduced, otherwise REDUCE(T,S) would have returned the null tree. By definition, the reduced form of T has an equivalent tree whose frontier is π. Let this equivalent tree be T'. The idea behind this proof is to construct a tree T'' which is equivalent

to T and which has π as its frontier. This is accomplished by undoing the template-matching which reduced T, but without undoing the equivalence transformations which were used during the template-matching. The process of applying a template has two stages; the pattern must be matched, possibly requiring an equivalence transformation, and then the replacement is made. Consider what happens when the templates are applied in reverse order from which they are applied during the ***merge and*** reduction algorithms. If the replacement is undone but the equivalence transformation used to match the pattern is left applied, the entire template-matching process can be run in reverse. The result is a Tree T" which is equivalent to the original Tree T but which has the same frontier as T'. This is the common frontier π, hence π ∈ CONSISTENT(T).

Next we show that π is also in CONSISTENT(T(U,S)), that is, the elements of S appear consecutively in π. By inspection of the ***merge*** templates, we see that after the REDUCE has completed there will be a node X in the tree such that:

a) the descendants of X include all of S, and
b) X is either a full P-node or a Q-node all of whose pertinent children are full and appear consecutively

From this it follows that all elements of S must appear consecutively in the frontier of any tree equivalent to the reduced tree.

Finally, we show that CONSISENT(T) ∩ CONSISTENT(T(U,S)) is contained in CONSISTENT(REDUCE(T,S)). That is, if a ***graph*** π is in the consistent set of the original tree and happens to have the elements of S appearing consecutively, then π is also in the consistent set of the reduced tree. Let T' be a tree equivalent to T and having π as its frontier.

The fact that S appears consecutively in π implies that for each PQ-tree, P, in T no node in the pertinent subtree of P, except the root, can have more than one partial child; the root may have at most two partial children ***when every node in S appears as a leaf of P, otherwise it may only have at most one (this is because it will need to be merged with other PQ-trees via the merge templates)***. Note also that after any partial node other than the root is matched, it becomes a Q-node; moreover, its sequence of children, examined from left to right (or possibly right to left), will consist of a sequence of full nodes followed by a sequence of empty nodes. These observations guarantee that each node matches one of the templates.

Next note that for each template, the fact that S is consecutive in the frontier of T' implies that the replacement can take place with no change to the frontier of the tree.

Thus π is in CONSISTENT(REDUCE(T',S)). Then since T' and T are equivalent, since template matching preserves equivalence, and ***since T' is well-formed***, π is also in CONSISTENT(REDUCE(T,S)).

QED.

With this result a simple inductive argument will show that the consistent set of a PQR-tree is the set of all trees, Z, whose vertices are exactly the set U and whose edges are subject to constraints S_1, ..., S_n where each constraint describes a subset of U that must ~~appear as~~be a path in Z.

We further claim that after all calls to REDUCE have completed the resulting PQR-tree will never have an R-collection R = {Q_1, ..., Q_z} where R is not RWF, $\cap_{i=1}^{z} Q_i = W$, and W has more than one child. This is captured by the following theorem:

3.2.2.3 Theorem: When executing MAIN(U, {S_1, ..., S_n}) if the PQR-tree being passed into VERIFY_R_COLLECTIONS is not NULL then for all R-collections R = {Q_1, ..., Q_z} of T the only way R can be malformed is when $\cap_{i=1}^{z} Q_i = W$, W has one child and W has more than one leaf.

Proof:

To prove this we consider without loss of generality the "last" Q-node, Q_z, in R that was formed. Q_z will have to be formed either through the application of one of Booth and Lueker's reduction templates or through one of our merge templates and neither of these will allow this to happen. We first suppose $W = \cap_{i=1}^{z} Q_i$ is the sub-Q-node {c_1, ..., c_k}, k>1 and let Q_z be the sub-Q-node {x, c_1, ..., c_k, y}[20]. Suppose without loss of generality we have Q_1 with sub-Q-node {x, c_1, ..., c_k} and Q_{z-1} with sub-Q-node {y, c_1, ..., c_k}. Now Q_z must be created using the merge templates but this is not possible because x and y occur on the same side of c_1, ..., c_k, due to Q_1 and Q_{z-1}. Thus, the algorithm will not allow the last Q-node in R to be created when W contains more than one child. Therefore, the only malformed R-collections in a PQR-tree passed into VERIFY_R_COLLECTIONS via MAIN will have W with exactly one child.

QED.

20 We form Q_z such that it only has one child on either side of the children of W without loss of generality as seen in the proof of **theorem 3.1.7 Claim#4**.

With this result we can now discuss the correctness of VERIFY_R_COLLECTIONS. This method proceeds by examining each node in the PQR-tree in a bottom-up way starting with each leaf. We do not consider the leaf itself because we need there to be more than one leaf that is shared among all Q-nodes in an R-collection. **Theorem 3.2.2.~~3~~2** tells us that we only need to consider the case when the overlap between the Q-nodes is a single child. This means that we can just check each node individually by examining all of its parents and making sure that they do not form a malformed R-collection. To do this we will form a graph out of the parents of a given node X, where each vertex in the graph corresponds to one of the parents and there is an edge between two vertices when the two corresponding parents have more than just X in common. We then examine the graph looking for odd length cycles and when this graph contains an induced odd length cycle the R-collection will contain an ordered odd sized subset that is not well-formed, and thus the PQR-tree contains a malformed R-collection. As such the consistent set of the PQR-tree will be empty by **theorem 3.1.6** and MAIN will appropriately return NULL.

Combining **theorem 3.2.2.~~3~~2** with **theorem 3.2.2.1** and **the** simple inductive argument using **theorem 3.2.2.~~2~~1** we have proven the following:

3.2.2.4 Corollary: if T = MAIN(U, {S_1, ..., S_n} then (CONSISTENT(T) = ∅ if and only if T = NULL) and CONSISTENT(T) = $\cap_{i=1}^{n}$ CONSISTENT(T(U,S_i)).

It is important to note that we have not provided a way of determining the consistent set of a given PQR-tree.

3.2.3 Efficiency

We will not concern ourselves with a detailed or particularly careful discussion of the efficiency of this approach to building a PQR-tree, since an optimal approach to building a PQR-tree is not the goal of this work. By noting the following:

- Booth and Lueker's reduction algorithm runs in linear time with respect to the size of the constraint being processed;
- Each call to the PQ-Merge method will run in linear time with respect to the height of the taller PQ-tree given as input;
- the fact that our reduction algorithm really only needs to concern itself with the nodes in

T that occur between S, the subset of T's leaves, and S's least common ancestors[21]; and

- We introduce at most one new PQ-tree per iteration.

We can see that each iteration of the first loop in our REDUCE method is linear with respect to the size of S, and thus the i^{th} iteration loop in MAIN will be O((number of PQ-trees) * $|S_i|$). Thus, the runtime of our PQR-tree construction algorithm (i.e. MAIN) is roughly O($(1*|S_1| + ... + n*|S_n|)$) which is O($n*(|S_1| + ... + |S_n|)$) before verifying the R-collections. The runtime of VERIFY_R_COLLECTIONS depends entirely on the number of edges within the each PQ-tree in the PQR-tree because as each node is processed we are only concerned with how many parents it has. In particular, this will be bounded by the sum of the size of the leaf set of each PQ-tree and this belongs to $O(|S_1| + ... + |S_n|)$ by construction. Therefore, the overall runtime of MAIN(U, $\{S_1, ..., S_n\}$) is O($n*(|S_1| + ... + |S_n|)$).

3.3 PQR-trees and Path Graphs

With PQR-trees defined and an algorithm to build a PQR-tree from a set of path constraints we can finally realize the goal of having a data structure that is to path graphs what PQ-trees are to interval graphs. We will use the following notation to discuss a graph G: V = $\{v_0, ..., v_{n-1}\}$ is G's vertex set, m is the number of edges in G, C = $\{c_0, ..., c_{\eta-1}\}$ is the maximal cliques of G, and S_i is the set of cliques incident with v_i for all i ∈ [0, n-1]. We will prove the following theorem:

3.3.1 Theorem: (PQR-tree) For a graph G the consistent set of the PQR-tree T = MAIN(C, $\{S_0, ..., S_{n-1}\}$) is exactly the set of all possible clique path intersection representations of G.

This theorem follows directly from **theorems 3.2.2.21** and **3.1.6**. These theorems tell us that the consistent set of T will contain all possible trees, $\{t_0, ..., t_{z-1}\}$, such that for all i ∈ [0,z-1], $V(t_i)$ = C and for all j ∈ [0,n-1] there exists a path P in t_i whose vertices are exactly S_j. Thus each of these trees satisfy the definition of a clique path intersection representation and CONSISTENT(T) contains all possible such trees. Therefore, CONSISTENT(T) will be all clique path intersection representations of G.

This theorem also provides us with the following corollary:

3.3.2 Corollary: (Path Graphs) A graph G is a path graph if and only if the consistent set of the PQR-tree T = (C, $\{S_0, ..., S_{n-1}\}$) is not empty.

21 By properties #2, and #3 from the definition of PQR-trees the number of these nodes will be linear with respect to S.

To prove this we just note that a graph G is a path graph if and only if there exists a path intersection representation of G such that each vertex in the intersection representation corresponds to a maximal clique in the graph **[10]**.

This gives us an approach for a recognition algorithm for path graphs. In particular, the algorithm presented in **section 3.2.1** will be able to tell us whether or not a particular graph is a path graph. However, it has not been written carefully to minimize runtime. Even with this consideration we can give a rough estimate of performance when attempting to recognize membership in the class of path graphs.

3.3.3 Corollary: Using the PQR-tree construction method membership in the class of path graphs can be determined in O(n (n+m)) time.*

Proof:

We do this using the following approach on a given graph G:

- We first check whether G is chordal (this is a linear time operation, i.e. O(n+m)) **[14]**;
- When G is not chordal it is not a path graph; and
- When it is chordal $|S_0| + ... + |S_{n-1}|$ will be linear with respect to the number of vertices and edges in G **[7]**, therefore, MAIN(C, $\{S_0, ..., S_{n-1}\}$) will run in time O(n*(n+m)).

QED.

This performance is quite comparable to the best known algorithm for recognition due to Schaffer **[13]**, which runs in O(η * (n+m)) time. Furthermore, the resulting PQR-tree will capture every clique intersection representation of the given path graph.

Theorem 3.3.1, and **corollaries 3.3.2** and **3.3.3** justify having path graphs as the motivating application for PQR-trees. **Theorem 3.3.1** and **corollary 3.3.2** demonstrate the relationship between PQR-trees and path graphs, giving us the result that a PQR-tree can be used to capture all path intersection representations of a graph. Furthermore, **Corollary 3.3.3** tells us that if we can improve the efficiency of the PQR-tree construction algorithm we automatically get a more efficient algorithm to recognize path graphs.

4 Future Work and Open Questions

The PQR-tree (see **section 3.1**) provides us with a powerful generalization of PQ-trees allowing us to capture all possible embeddings of a set U onto a tree such that for every S_i in a given family of subsets $\{S_0, ..., S_{z-1}\}$ of U, S_i is a path in that tree. Furthermore, given U and a family of subsets of U we have developed an algorithm (see **section 3.2**) that constructs a PQR-tree when such an embedding exists and returns NULL when no such embedding exists. This data structure and algorithm have a direct application to path graphs (see **section 3.3**) providing a new recognition algorithm for path graphs which also produces a PQR-tree capturing all path intersection representations of the given graph[22].

This initial work in developing PQR-trees provides a foundation for many avenues of potential future work. In **section 4.1** we discuss some future work directly involving PQR-trees and in **section 4.2** we will discuss some of the observations about path graphs that have arisen through the development of PQR-trees and some corresponding open questions.

4.1 Future Work

In this section we will discuss some of the avenues of potential future work relating to PQR-trees and their application to path graphs. We will not consider applications of PQR-trees outside of graph theory, but we expect there to be many of these due to the fact PQ-trees have seen many applications throughout mathematics (see **[1]** and **[9]** for the initial applications of PQ-trees).

There are still some algorithms relating to PQR-trees that we have not developed. Some of these include:

- Given a PQR-tree, T, with CONSISTENT(T) $\neq \emptyset$, output a PQR-tree T' such that T' $\equiv$ T and T' is TWF.
- Given a PQR-tree, T, with CONSISTENT(T) $\neq \emptyset$, output the size of the set CONSISTENT(T).
- Given a PQR-tree, T, with CONSISTENT(T) $\neq \emptyset$, output the set CONSISTENT(T).

We have also yet to optimize the algorithm presented in **section 3.2.1**. The version presented in **section 3.2.1** was not carefully designed to be optimal in terms of efficiency and given a set U

22 When the given graph is not a path graph the algorithm will produce a NULL PQR-tree since the graph will not have any path intersection representations.

and a family of subsets $\{S_0, ..., S_{z-1}\}$ of U will run in $O(z * (|S_0| + ... + |S_{z-1}|))$ time (see **section 3.2.2**). The efficiency of this algorithm is strongly tied to the efficiency of recognizing path graphs since we can use it to produce an algorithm to recognize path graphs in O(n*m) time (see **section 3.3**). If we are able improve its runtime to $O(|S_0| + ... + |S_{z-1}|)$ then we will have improved the runtime of our recognition algorithm to O(n+m) which is better than the current best known recognition algorithm (due to Schaffer **[13]** running in O(n*m) time).

It has been shown that isomorphism between interval graphs can be done by comparing their corresponding PQ-trees and that this can be done in linear time with respect to the sizes of these graphs **[9]**. This leads to the following:

Open Questions: Can PQR-trees be used to decide isomorphism between two path graphs? If so, what efficiency can be achieved? (Note: this is of particular interest due to the isomorphism completeness of path graphs as seen in **section 2.1**)

We can also consider the role that malformed R-collections play in determining membership in the class of path graphs. In particular, if we consider the graphs that would not be rejected before reaching the verification of R-collections, but would be rejected by the verification we get some interesting results. We claim that these are the graphs that contain an induced subgraph which is an odd sized Sun where there are multiple vertices whose neighbourhood is the centre (clique) of the Sun (similar to the graph in **figure 4.6)**. When we combine this with **claim 4.2.1.2**, we can see that by changing the condition on R-collections being not well-formed to no longer constrain the number of leaves of "W" we would then forbid odd sized Suns as induced subgraphs, and as such would then be able to determine membership in the class of directed path graphs. Using our algorithm for this purpose would not be particularly interesting in terms of performance since there already exists a linear time algorithm to determine membership in the class of directed path graphs **[4]**.

We can also generalize the approach we have taken in building a PQR-tree for path graphs to similar graph classes. In particular, a collection of PQ-trees has the potential to capture any family of graphs where each vertex can be represented by a path and the union of these paths is subject to a certain constraint (i.e. in the case of path graphs this constraint would be that their union forms a tree).

4.2 Asteroidal Triples, the Structure of Path Graphs, and Open Questions

This section will discuss some interesting observations and results regarding path graphs that have been seen through the development of the PQR-tree. Furthermore, we mention some open questions arising from these observations and results. In particular, we explore the role Asteroidal Triples play in the structure of path graphs and operations under which path graphs are both closed and not closed. The operations considered have been examined with the goal of developing a basis for a set of transformations that will allow the creation of an arbitrary path graph. Asteroidal triples are defined in **section 2.1.1**. ATs are an interesting structural element in interval, path, and chordal graphs since interval graphs are the graphs which are chordal and AT-Free (see **theorem 2.1.1.3**), and interval graphs $\subset$ directed path graphs $\subset$ path graphs $\subset$ chordal graphs (see **theorem 2.1.1.1**).

A graph G is a "Minimal AT graph" when G contains an AT such that the removal of any vertex will make the graph AT-free. The first subsection will address minimal chordal ATs and their membership in the set of path graphs. The next subsection will address the effect of adding specific kinds of vertices to path graphs and whether or not the resulting graph is still a path graph. Finally, the third subsection will examine how to use path intersection representations of path graphs to create new path graphs.

4.2.1 Minimal Chordal Asteroidal Triples and Path Graphs

The set of minimal ATs has been well studied by Kohler **[5]** and provides a set of graphs which are known not to be interval, since interval graphs are AT-free, and may or may not be path graphs. The first subset of these graphs that we can eliminate as being path graphs are those that are not chordal, since path graphs are strictly contained within the chordal graphs. This leaves the following minimal chordal ATs that may be path graphs: the “Extended Claw”, the “Net”, the “Generalized Net”, the “3-Sun”, the “Generalized 3-Sun”, and the “Umbrella” (see **figure 4.1**). From these graphs all but the Generalized 3-Sun where $i \geq 3$ are path graphs. This can be seen through the following intersection representations of these graphs (see **figure 4.2**).

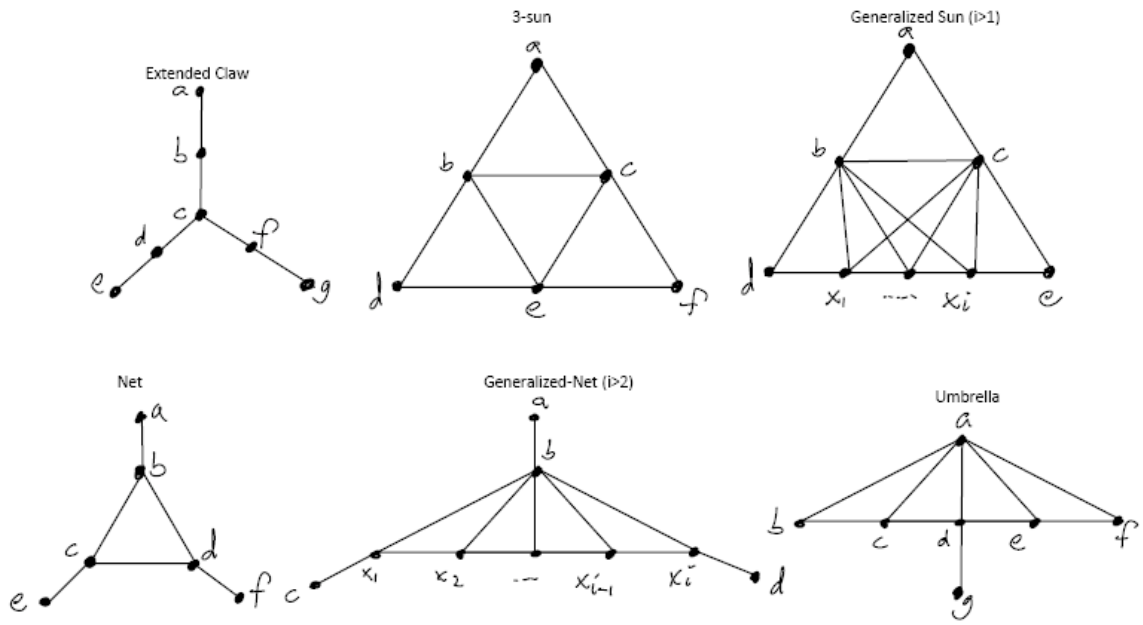


Figure 4.1: Minimal Chordal Asteroidal Triples

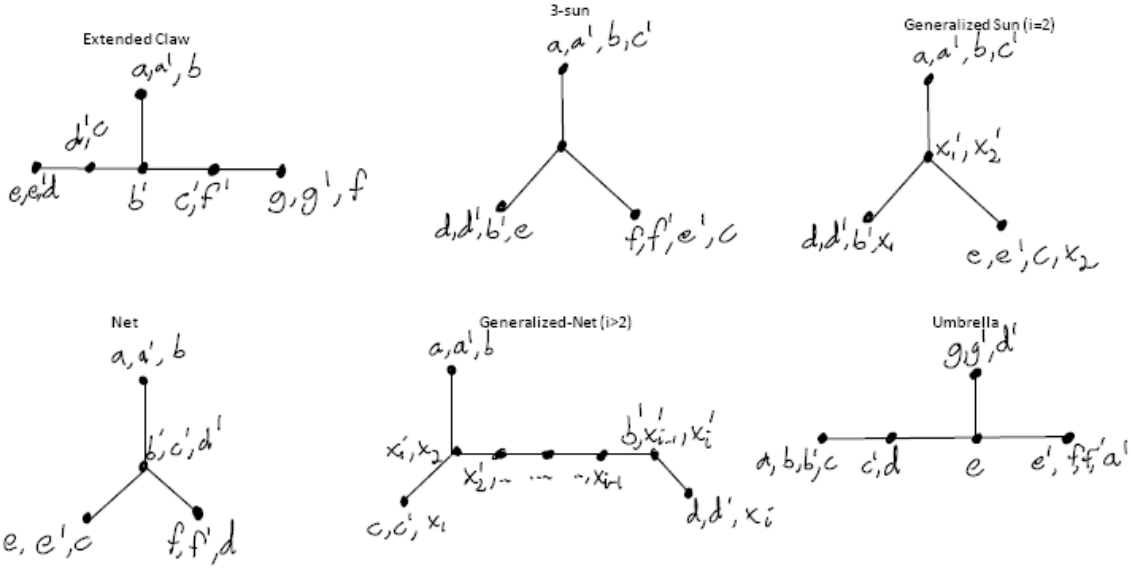


Figure 4.2: Path Intersection Representations of the minimal chordal ATs (the path corresponding to a vertex, v, from the original graph will consist of every node starting from v and ending at v' in that graph's corresponding representation).

These intersection representations were created via Monma and Wei's Clique Tree theorem (see section **2.1.3**). Using this theorem we prove that the generalized 3-Sun (i>2) is not a path graph (i.e. has no clique path intersection representation).

4.2.1.1 Lemma: If a graph G is a generalized 3-Sun (i>2) then G is not a path graph. (this is confirmed by the proposed FISC of path graphs in section 2.1.5)

Proof:

Let the cliques of G (with vertex labels as in **figure 4.1** above) be: $C_0 = (b,d,x_1)$, $C_j = (b,c,x_j, x_{j+1})$ for all $j \in [1,i-1]$, $C_i = (c,e,x_i)$, and $A = (a,b,c)$. For $j \in [1,i]$ each x_j is only contained in two cliques, C_{j-1} and C_j, therefore, every clique path intersection representation must have the edge $C_{j-1}C_j$. Since vertex b is incident with cliques C_0, ..., C_{i-1}, and A, therefore A must be adjacent to either C_0 or C_{i-1}. Also vertex c is incident with cliques C_1, ..., C_i, and A, therefore A must be adjacent to either C_i or C_1. But now, any combination of the adjacency of A required by b and the adjacency of A required by c will create a cycle in the intersection representation. Therefore, the representation must contain a cycle when each vertex is represented by a path.

QED.

We notice that the Generalized 3-Sun appears in Panda's forbidden characterization of directed path graphs (see graph A_6 from **figure 2.1** in **section 2.1.5**). This leads to the following question: what graphs from Panda's FISC of directed path graphs are path graphs? We first notice that the set of graphs described by A_{15} is not chordal and thus not part of the path graphs. We claim that the graphs A_1, A_2, A_3, A_4, A_5, A_8, A_9, A_{10}, $A_{11}(k>1)$, $A_{12}(k>1)$, and $A_{14}(k>1)$ are not path graphs (this can be seen by applying a similar argument to the one applied to A_6 above and some of these will be revisited in **section 4.2**). However, A_7 is the 3-Sun which, as we have already seen in **figure 4.2**, is a path graph. Also, $A_{13}(k>1)$ is the set of (2k+1)-Suns (or "odd sized Suns"), k>1, which are also path graphs (this can be seen by extending the intersection representation for the 3-Sun from **figure 4.2**). The graphs A_3, $A_{11}(k>1)$, $A_{12}(k>1)$, and $A_{14}(k>1)$ are missing from Tondato et al.'s proposed FISC of path graphs (see section **2.1.5**) meaning that their FISC of path graphs is incomplete. This also leads to the following claim regarding the FISC of path graphs:

4.2.1.2 Claim: If a graph G is a member of the FISC of path graphs and $G \notin \{A_1, A_2, A_3, A_4, A_5, A_8, A_9, A_{10}, A_{11}(k>1), A_{12}(k>1), A_{14}(k>1)\}$ then G contains an induced subgraph $H \in \{A_7, A_{13}(k>1)\}$ (i.e. H is an odd sized Sun)

This means that the only difference between path graphs and directed path graphs is the fact that the directed path graphs don't allow the odd sized Suns as induced subgraphs and that the path graphs do allow the odd sized suns as induced subgraphs but possibly only in some limited

capacity.

4.2.2 The Effect of Adding Vertices to Path Graphs

Now that we have established a set of graphs that are path graphs we can examine what kinds of operations can be performed on these graphs that allow the graphs to remain chordal, but not remain path graphs. We will call an operation that is applied to a path graph in which the result is a non-path chordal graph a "Contamination." In particular, we explore how we can contaminate path graphs using vertex additions and how we can add vertices to path graphs without contaminating them.

4.2.2.1 Asteroidal Triple Theorem for Path Graphs

First, we consider how a single vertex can contaminate path graph containing an AT. Notice that we have already seen an AT-free path graph (i.e. an interval graph) that can be contaminated by adding a single vertex (consider any proper induced subgraph of size n-1 of a generalized 3-Sun with $i \geq 3$). So, such contaminations are possible, and thus worth exploring in more detail. Considering the path intersection representations of the minimal ATs that are path graphs (see **figures 4.1, 4.2**), we will show that when adding a new vertex which is adjacent to all three vertices in the AT, that vertex's representation in the intersection graph must be a tree. In fact, this is not only true for the minimal ATs but for all graphs containing an AT. Thus, any graph containing an AT and a vertex which is universal to that AT cannot be a path graph. To prove this we first prove the following:

4.2.2.1.1 Lemma: If a graph G is a path graph and $v_1 \in V(G)$, then $G[N(v_1)]$ is an interval graph

Proof:

Let $I = \{S_1, ..., S_n\}$ be a path intersection representation of G and S_1 be the set from I corresponding to v_1. Consider the following intersection representation: $J = \{S_i \cap S_1 : i > 1, S_i \cap S_1 \neq \emptyset\}$. J is now a collection of subpaths of S_1 and thus J represents an interval graph. Furthermore, for the sets S_j, corresponding to vertices in $N(v_1)$, $S_j \cap S_1 \neq \emptyset$ and for the sets S_k, corresponding to vertices in $V(G) - N[v_1]$, $S_k \cap S_1 = \emptyset$.

We now claim that J is an intersection representation for $G[N(v_1)]$. Suppose there exists an edge in $G[N(v_1)]$ that is not represented by J. This implies that there exist v_p and v_q in $N(v_1)$ such that v_p and v_q are adjacent (i.e. $S_p \cap S_q \neq \emptyset$) and $(S_p \cap S_1) \cap (S_q \cap S_1) = \emptyset$ (or

equivalently $S_1 \cap (S_q \cap S_p) = \emptyset$). However, we know that $(S_q \cap S_1) \neq \emptyset$, $(S_p \cap S_1) \neq \emptyset$ and S_q, S_1, and S_p are paths. So $(S_q \cup S_p \cup S_1)$ forms a cycle, contradicting the fact that the union of all $S_i \in I$ forms a tree. Therefore, J is an intersection representation for $G[N(v_1)]$, meaning that $G[N(v_1)]$ must be an interval graph.

QED.

With this result proving the claim is now fairly straightforward.

4.2.2.1.2 Corollary: (AT Theorem for Path Graphs) Given a graph G with AT, A = {a,b,c}, and x in V(G) such that x is universal to A, then G is not a path graph.

Proof:

Suppose G is not chordal, then G is not a path graph. Therefore, we only need to consider graphs which are chordal.

Consider a minimal induced subgraph, H, of G such that A is still an AT in H. Thus x does not belong to H, since H is minimal. Suppose that x is not universal to H. This means there exists a vertex z in H such that z is not adjacent to x. Notice that z must be required by one of the AT paths in H. Without loss of generality we now suppose the (a,b)-path, P, contains z. Consider the neighbours of x on P that are closest to z, but not equal to z. We call these vertices u and v, such that u is on the (a,z)-path and v is on the (b,z)-path (these are well-defined because of a and b). Let Q be the subpath of P starting at u and ending at v, but now $G[Q \cup \{x\}]$ is an induced cycle of size larger than 3. This contradicts the fact that G is a chordal graph, and implies that x is universal to H. Since H contains an AT it is not an interval graph (see **theorem 2.1.1.3**) which means that N(x) is not an interval graph, therefore G is not a path graph by **lemma 4.2.2.1.1**.

QED.

Applying the AT Theorem for path graphs to Panda's FISC of directed path graphs (see **section 2.1.5**) we see that A_1, A_5, A_8, A_9, A_{10} are actually the extended claw, the generalized net, the net, and the umbrella, all with a universal vertex and therefore none of these can be path graphs.

In a further corollary to this we notice the following:

4.2.2.1.3 Corollary: Path graphs are not closed under vertex replacement by path graphs.

Proof:

Consider the graph G = Extended Claw, which is a path graph (see **section 4.1**) and has an AT. Replacing any vertex with at least one neighbour in a path graph, say $H = K_2$, with G

creates a graph, H', that contains a vertex universal to an AT. Therefore, H' is not a path graph.

QED.

Open Questions: This last corollary leads to some interesting questions regarding vertex replacement. First, what family of graphs is obtained by starting with a path graph and replacing its vertices with path graphs? Second, starting with a path graph, G, what graphs can we replace the vertices of G with, such that the new graph will still be a path graph? Also, a modification to the second question could be considering only special kinds of vertices in G, in particular simplicial vertices and what these could be replaced with while the resulting graph is a path graph. It is interesting to note that using vertex replacement, a non-chordal graph is easily created from interval graphs. For example, starting with two P_3's and replacing the middle of one P_3 with the other P_3 we create a 4-wheel which is not chordal (see **figure 4.3** below).

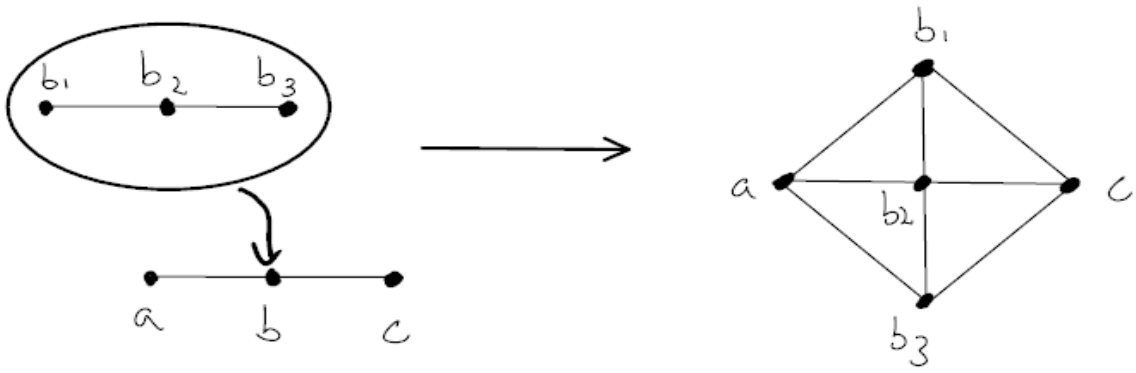


Figure 4.3: P_3's A = (a,b,c) and B = (b_1, b_2, b_3), replacing vertex b from A with B creates the non-chordal 4-wheel on the right

4.2.2.2 Stars and Twins

We now consider some non-contaminating vertex additions. To do this we first need a few definitions: a "star vertex" in a graph G is a vertex x such that N(x) is an independent set, in particular G[N[x]] is a star, a "true twin vertex" in a graph G is a vertex x such that there exists vertex y, different from x, and N[x] = N[y], and a "false twin vertex" in a graph G is a vertex x such that there exists vertex y, not adjacent to x, different from x, and N(x) = N(y). The follow set of lemmas provide some interesting insights into the structure of path graphs.

4.2.2.2.1 Lemma: (Star Lemma) Given a connected chordal graph G with a star vertex x, G is

a path graph if and only if G - {x} is a path graph.

Proof:

We first note that when G is a path graph, G – {x} for any vertex x of G is also a path graph since path graphs are hereditary.

We now consider the graph H = G – {x} for some star vertex x of G where H is a path graph. We denote N(x) = $\{v_1, ..., v_k\}$ and the connected components of H to be $cc_1, ..., cc_k$ such that v_i belongs to cc_i. It is important to note that since G is chordal each cc_i must be distinct, otherwise we would have a induced cycle of length greater than 3 (i.e. $cc_i \neq cc_j$ for all i,j in [1,k] and $i \neq j$). Since H is a path graph, therefore each connected component is also a path graph. Letting $I(cc_i)$ be the path intersection representation of cc_i and $P(v_i)$ be the path representing v_i, we create the path intersection of G as follows. For each v_i add a new vertex x_i to $P(v_i)$ and extend one of the ends of $P(v_i)$ to include x_i by adding the appropriate edge. Now we construct $P(x) = \{x_i : i = 1, .., k$ and x_j adjacent to $x_{j+1}, 1 \leq j < k\}$ and create a path intersection representation of G, $I(G)=\{P(cc_i) : i = 1, .., k\} \cup \{P(x)\}$.

QED.

It can be shown that the Star Lemma also applies to directed path graphs simply by adding the appropriate directions to the new edges added. An immediate corollary to the Star Lemma is the following:

4.2.2.2.2 Corollary: All trees are (directed) path graphs.

Proof:

(by induction on the number of vertices in the Tree)

When T is a single vertex it is trivially a (directed) path graph. We now assume for all Trees T where |V(T)| < k, T is a (directed) path graph. Now let T be a tree such that V(T) = k and choose a vertex v from T arbitrarily. Notice that N(v) must be an independent set since T is a tree. Therefore, T is a (directed) path graph if and only if T' = T – {v} is a (directed) path graph by the Star Lemma. Applying our assumption to T' we can see that T' is a (directed) path graph, since |V(T')| = |V(T)| –1 < k. Thus, T must also be a (directed) path graph.

Therefore, all Trees are (directed) path graphs.

QED.

A side note to **corollary 4.2.2.2.2** is that it implies the Extended Claw is a path graph. Now that

we know all trees are path graphs, it raises the question of whether k-trees are path graphs for fixed k. However,it turns out that the Extended Claw with a universal vertex (**figure 4.4** below) is a 2-tree, thus even two trees are not completely contained within path graphs. Furthermore, using the same construction we see that for any fixed k>1, a there is a k-tree that contains the Extended Claw with a universal vertex.

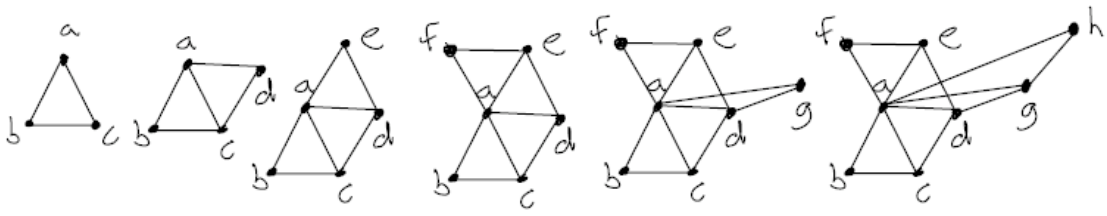


Figure 4.4: Construction of a 2-tree which is an Extended Claw with a universal vertex (a)

Now that we have a lemma for trees, we look at another basic graph family, the cliques. This brings us to the True Twin Lemma:

4.2.2.2.3 Lemma: (True Twin Lemma) Path graphs are closed under the addition and deletion of true twins

Proof:

Once again deletion is ok due to the fact that path graphs are hereditary.

Let I be a path intersection representation of a path graph and x be the vertex we are creating a true twin from, such that S_x is the path representation of x in I. We now add S_y to I where $S_y = S_x$ and we call the vertex that S_y represents y. I is clearly still a path intersection representation. Also, $S_y = S_x$ implies N[y] = N[x] in the graph I represents. Therefore, we have created a new path graph from an existing path graph such that the only difference is a true twin of vertex x has been added to the original graph.

QED.

Similarly to how the Star Lemma demonstrated the fact that the Extended Claw is a path graph, combining the True Twin and Star lemmas we can see that the Net is a path graph. Also, the True Twin Lemma provides some interesting corollaries:

4.2.2.2.4 Corollary: Path graphs are closed under vertex replacement by cliques.

Proof:

Vertex replacement by cliques can be thought of as a sequence of additions of True Twins.

To replace vertex x in a path graph by K_i, $i > 1$ requires the addition of i-1 True Twins of x and path graphs are closed under the addition of True Twins by the True Twin Lemma.

QED.

A side note about **corollary 4.2.2.2.4** is that it implies that all cliques are path graphs, which we already knew since interval graphs are a proper subset of path graphs **[10]**.

4.2.2.2.5 Corollary: G is a path graph if and only if H is a path graph, where H is the maximal induced subgraph of G such that H contains no true twins.

Proof:

Follows directly from the True Twin Lemma.

QED.

Now that we have examined True Twins, we look into False Twins. First, we note that if we try to add the false twin of a vertex whose neighbourhood is not a clique to any graph the result will be a graph containing an induced 4-cycle and thus is not chordal. Therefore, we will only consider false twins of simplicial vertices. It turns out that in general we cannot add simplicial false twins to path graphs, but we can do so for interval and directed path graphs.

4.2.2.2.6 Lemma: (False Twin Lemma for interval graphs) interval graphs are closed under the addition of a false twin of vertex x when N(x) is a clique.

Proof:

Let G be an interval graph with vertex x. G is clearly chordal, therefore the addition of a false twin, x', of x cannot introduce an AT (since N(x') = N(x)) and since N(x) is a clique the graph will still be chordal. Therefore, adding a false twin of x to G will create a new graph that is both chordal and AT-free, and therefore interval.

QED.

(Alternate Proof via intersection representations) :

Let I be a clique interval intersection representation of an interval graph G **[10]** and for each vertex z in G let S_z be the set representing z in I. Since N(x) is a clique, this means that N[x] must be a maximal clique and thus represented by a single vertex, c, in I. Furthermore, $S_x = \{c\}$ and for all vertices v, in N(x), S_v must contain c. To create the false twin of x, call this vertex x', we will create a new intersection representation I', where I' is a copy of I with the following modifications: an edge incident with vertex c is subdivided calling the new vertex

c'; vertex c' is then added to each S_v where v belongs to N(x), and finally a new set $S_{x'} = \{c'\}$ is added. Notice that each modified path has had a single edge subdivided, meaning that it is still a path, and the union of all sets in I' is a path since it has also just had a single edge subdivided. Furthermore, in I' N(x') = N(x) since all paths that contained c in I, except S_x, contain c' in I', no other paths in I' contain c', and $S_{x'} = \{c'\}$. Therefore, I' is an interval intersection representation and the graph it represents is precisely G with a new simplicial false twin of x.

QED.

A similar argument will work for False Twins and directed path graphs. Consider the following:

4.2.2.2.7 Lemma: (False Twin Lemma for directed path graphs) Directed path graphs are closed under the addition of a false twin of vertex x when N(x) is a clique.

Proof:

Let I be a clique directed path intersection representation of a directed path graph G **[10]** and for each vertex z in G let S_z be the set representing z in I. Since N(x) is a clique, this means that N[x] must be a maximal clique and thus represented by a single vertex, c, in I. Furthermore, $S_x = \{c\}$ and for all vertices v, in N(x), S_v must contain c. To create the false twin of x, call this vertex x', we will create a new intersection representation I' (see **figure 4.5**), where I' is a copy of I with the following modifications: split vertex c into two vertices c_1 and c_2, add a directed edge from c_1 to c_2, for all vertices v such that vc is an edge in I we add the directed edge vc_1 to I', add c_1 and c_2 to S_v, and remove c from S_v, similarly for all vertices u such that cu is an edge in I we add the directed edge c_2u to I', add c_1 and c_2 to S_u, and remove c from S_u, then we set $S_x = \{c_1\}$, and finally a new set $S_{x'} = \{c_2\}$ is added. Notice that each modified path has had a single vertex inserted, meaning that it is still a path, and the union of all sets in I' is a directed tree. Furthermore, in I' N(x') = N(x) since all paths that contained c in I, except S_x, contain c_2 in I', no other paths in I' contain c', and $S_{x'} = \{c_2\}$. Also, the adjacency of x has not changed, since all paths that contained c in I now contain c_1 in I', no other paths in I' contain c_1, and $S_x = \{c_1\}$. Therefore, I' is a directed path intersection representation and the graph it represents is precisely G with a new simplicial false twin of x.

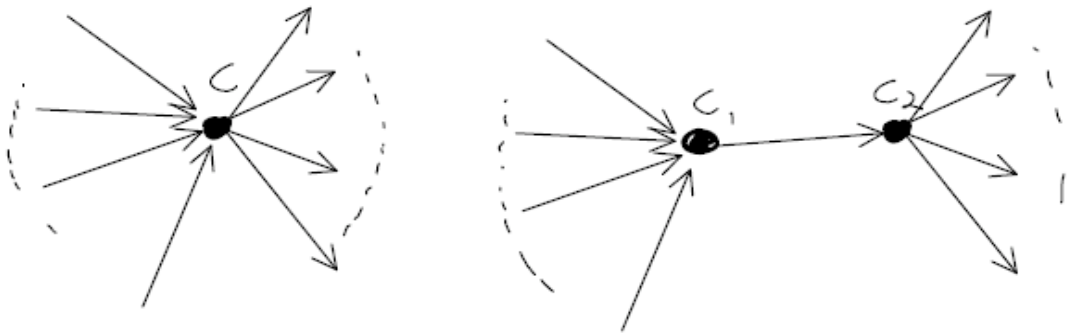


Figure 4.5: Intersection representation I and vertex c (left), Intersection representation I' and vertices c_1, c_2 (right)

QED.

However, when we attempt to extend the argument to path graphs it breaks down and in general we cannot add simplicial false twins to path graphs. In the following graph we have a 3-Sun with a simplicial vertex, x, added to its centre clique (see **figure 4.6**).

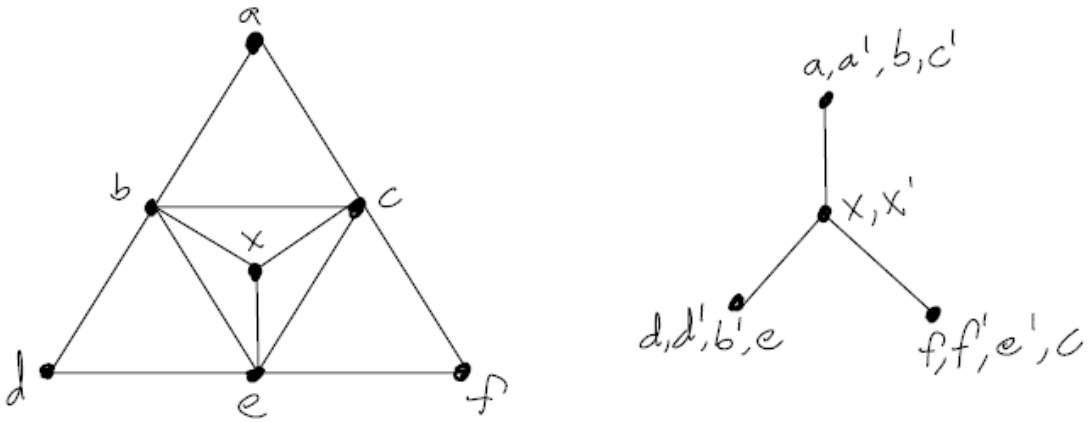


Figure 4.6: 3-Sun with a simplicial vertex added to its centre clique (left), clique path intersection representation (right)

If we add a false twin, y, of vertex x to this graph it will no longer be a path graph. This is because vertex y will introduce a new clique (b,c,e,y) into the graph and in order to add this clique to the intersection representation it must be a vertex subdividing one of the edges in the intersection representation in **figure 4.6**[23]. Furthermore, no matter which edge this new vertex is placed on, it will not be possible for the representation to remain a path intersection representation. In particular, this graph with vertex y can be seen as graph **4** in the FISC of path

23 We can say this because this is the only clique intersection representation of this graph

graphs proposed by Tondato et al. in **section 2.1.5 figure 2.3**.

Open Question: How can we characterize the vertices in path graphs that permit the creation of False Twins? (Note: they must be simplicial otherwise the resulting graph will not be chordal). We propose the following conjecture to answer this question:

4.2.2.2.8 Conjecture: Path graphs are closed under the addition of false twins to simplicial vertices whose neighbouring clique is not the "centre" of an odd-sized-Sun.

With **lemmas 4.2.2.2.6** and **4.2.2.2.7** we have the following corollaries:

4.2.2.2.9 Corollary: G is an interval graph if and only if H is a interval graph, where H is the maximal induced subgraph of G such that H contains no simplicial false twins.

Proof:

Follows directly from **lemma 4.2.2.2.6** and the fact that interval graphs are hereditary.

QED.

4.2.2.2.10 Corollary: G is a directed path graph if and only if H is a directed path graph, where H is the maximal induced subgraph of G such that H contains no simplicial false twins.

Proof:

Follows directly from **lemma 4.2.2.2.7** and the fact that directed path graphs are hereditary.

QED.

These lemmas and their corollaries provide some interesting potential implications to recognizing membership in the set of path graphs since the pre-processing deletion of these vertices does not affect whether or not the original graph is a path graph.

Open Questions: Can we improve on or simplify existing recognition algorithms for path graphs by applying these lemmas as a pre-processing step? What does the subset of path graphs which are not directed path graphs that contain no Star vertices and no True Twins look like and what do their path intersection representations look like? Another question that we can think about is whether we can classify certain simplicial vertices as "Directed Path Simplicial" so that we can still add false twins of them without contaminating the graph (similarly to c**onjecture 4.2.2.2.8**).

4.2.3 Growing path graphs through their intersection representations

We now examine ways to combine two path graphs to create a new path graph and how we can add edges to an existing path graph to create a new path graph, with the potential goal of creating a set of operations and initial graphs that allow us to build an arbitrary path graph. We start by looking at a way to glue two path intersection representations together using a new

vertex.

4.2.3.1 Observation: (Glue#1) Given path graphs G and H with path intersection representations I and J respectively. If P_G is a path in the tree $T_G=\cup_{S\in I}S$ and P_H is a path in the tree $T_H=\cup_{S\in J}S$ (Note: P_H and P_G may or may not represent vertices in H and G respectively) then K is a path intersection representation where:
K = is the set consisting of all sets from I and J, and S_x ={$P_G \cup P_H$: such that an edge is added from one end of P_G to one end of P_H}.

Proof:

K will be a path intersection representation since all we have done is add a single edge between T_G and T_H and add the new set S_x which is a path.

QED.

We can also "Glue" two path graphs together without adding a new vertex by the path representing a vertex in the first graph into the intersection representation of the second graph.

4.2.3.2 Observation: (Glue#2) Given path graphs G and H with path intersection representations I and J respectively. If v is a vertex in G, represented by S_v in I and P_H is a path in the tree $T_H=\cup_{S\in J}S$ (Note: P_H may or may not represent a vertex in H) then K is a path intersection representation where:
K = all the sets from I and J except S_v is replaced by $S_v' = \{S_v \cup P_H$: such that an edge is added from one end of S_v to one end of P_H}.

Proof:

I_F will be a path intersection representation since all we have done is add a single edge between T_G and T_H, and extend the path S_v along this edge into T_H.

QED.

In a similar fashion to Glue#2 we can add edges to a given path graph. We do this by selecting a path intersection representation, then we extend one path in it thus potentially increasing the number of edges within a path graph to create a new path graph.

4.2.3.3 Observation: (Edge Additions) Given a path graph G and a path intersection representation I for G. If v is a vertex of G, represented by S_v in I, and x is a vertex in the tree $T_G=\cup_{S\in I}S$ adjacent to one of the ends of S_v then I' is a path intersection representation where I' = all the sets from I except S_v is replaced by S_v' = S_v union x.

Proof:

I' will be a path intersection representation since all we have done is extend the set S_v to be a longer path.

QED.

The edge additions observation provides a starting point for determining all possible sets of

edges that can be added to a path graph without contaminating it. If we were to consider all possible intersection representations at once and the possible ways that each set within them can be stretched or shrunken we would be able to determine which edges could be added or removed from a path graph without contaminating it. As such, it would be interesting to revisit these in the context of PQR-trees since PQR-trees can represent all possible path intersection representations of a path graph (see **Chapter 3**).

Open Questions: Do these observations form a basis for building an arbitrary path intersection representation? It would be interesting to determine equivalent lemmas in terms of the vertices and edges of a path graph rather than the intersection representation.

5 References

[0] L. Babel, I.N. Ponomarenko and G. Tinhofer, “The isomorphism problem for directed path graphs and for rooted directed path graphs”, Journal of Algorithms, 21 (1996), 542–546.

[1] 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, 335-379.

[2] E. Dahlhaus, G. Bailey, “Recognition of Path Graphs in Linear Time” The Fifth Italian Conference on Theoretical Computer Science" (Revello, 1995) 201-210, World Sci. Publishing, River Edge, NJ, 1996

[3] E. Dahlhaus, Private Communications, 2007.

[4] P. Dietz, “Intersection Graph Algorithms”, Ph.D. thesis, Comp. Sci. Dept., Cornell University. Ithaka, NY, USA (1984)

[5] E. Kohler, “Graphs without Asteroidal Triples” Ph.D. thesis, Dept. of Mathematics, Technical University of Berlin. Berlin, Germany (1999).

[6] F. Gavril, “A Recognition Algorithm for the Intersection of Graphs of Paths in Trees”, Discrete Mathematics, 23 (1978), 211-227

[7] F. Gavril, “the Intersection Graphs of Subtrees of Trees are Exactly the Chordal Graphs”, Journal of Combinatorial Theory Series B 16, 1974, 47-56.

[8] C. Lekkerkerker and J. Boland, “Representation of a finite graph by a set of intervals on a real line”, Fund. Math., 51, (1962), 45-64

[9] G. Lueker and K. Booth, “A Linear Time Algorithm for Deciding Interval Graph Isomorphism”, Journal of the Association for Computing Machinery, Vol 26, No 2, April 1979

[10] C. Monma and V.Wei. “Intersection graphs of paths in a tree.” J. Combin. Theory B. 41. (1986) 141–181.

[11] B.S. Panda, “The forbidden subgraph characterization of directed vertex graphs.” Discrete Mathematics 196 (1999), 239–256.

[12] D. Rose, R. Tarjan, G. Lueker, “Algorithmic aspects of vertex elimination orderings on graphs” SIAM J. Comput., 5 (1976), 266–283.

[13] A. Schaffer, “A faster algorithm to recognize undirected path graphs”, Discrete Applied Mathematics, 43 (1993), 261-295.

[14] R. Tarjan, M. Yannakakis, “Simple linear time algorithms to test the chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs”, SIAM J. Comput. Vol. 13, No. 3, August 1984.

[15] S. Tondato, M. Gutierrez, J. Szwarcfiter, “A forbidden subgraph characterization of path graphs.” Electronic Notes in Discrete Mathematics, 19 (2005), 281–287.

[16] S. Tondato, J. Szwarcfiter. Private Communications, 2007.

Appendix: Booth and Lueker's reduction templates

Included here are the templates from Booth and Lueker's Reduction algorithm [**1**]. While applying these templates via our algorithm we will have to be careful to propagate the changes made to the PQ-tree being operated on to all PQ-trees that share the nodes involved in the current operation. To accomplish this we will store every maximal collection, C, of overlapping Q-nodes[24] as a single "R-node", where an R-node, R, is the union of all Q-nodes in C such that each Q-node from C is a "sub-Q-node" of R (i.e. R can be thought of as a "tree" which is the union of "paths" (Q-nodes in C)). This will mean that every Q-node in a PQR-tree will be represented by referencing into an R-node[25]. We first discuss how R-nodes change with respect to each template.

We first give the templates for P-nodes (note: P0, P1, and P2 do not affect R-nodes).

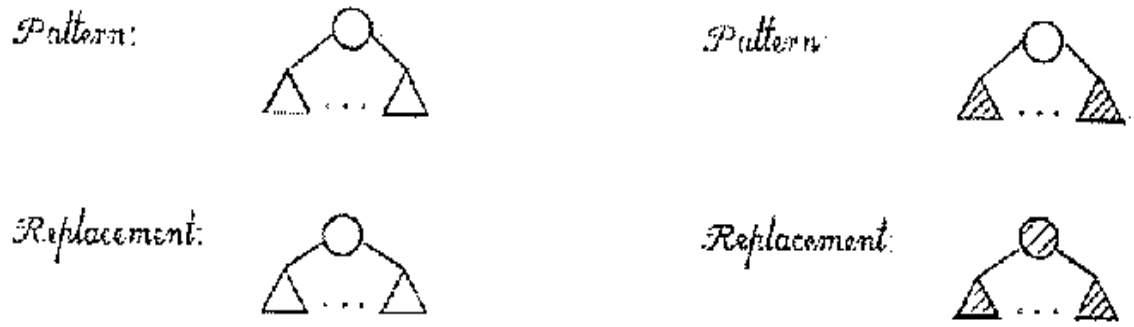


FIG. 7. Easy cases for P-nodes—Template *P0* (left) and Template *P1* (right).

24 By overlapping we mean for every Q-node, A, in this collection there exists another Q-node, B, in the collection such that A and B have a sub-Q-node in common.

25 A R-node will be unique for a given Q-node.

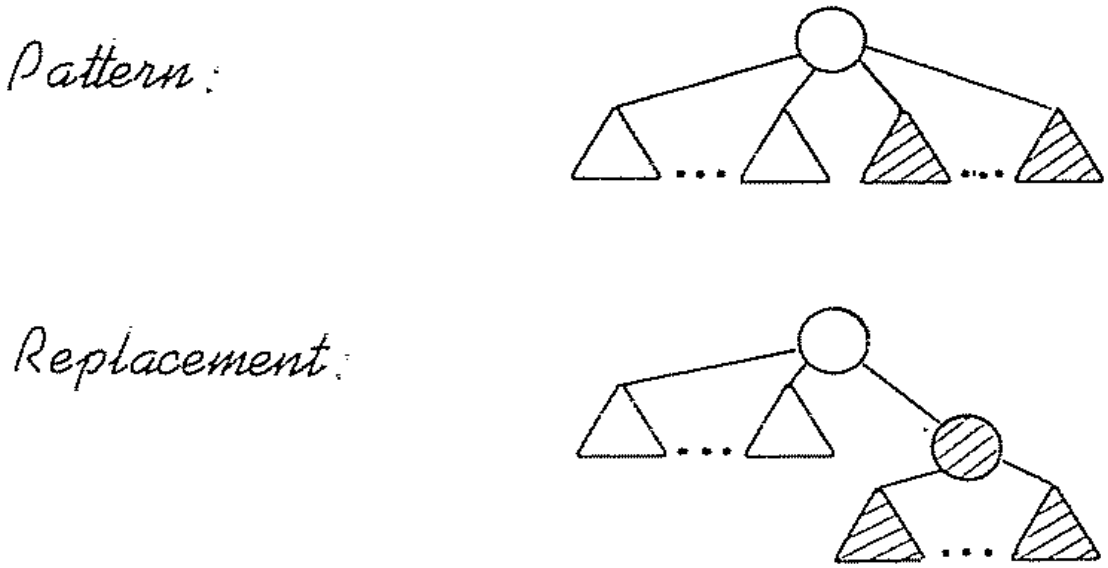


FIG. 8. Template $P2$ for ROOT(T, S) when it is a P-node.

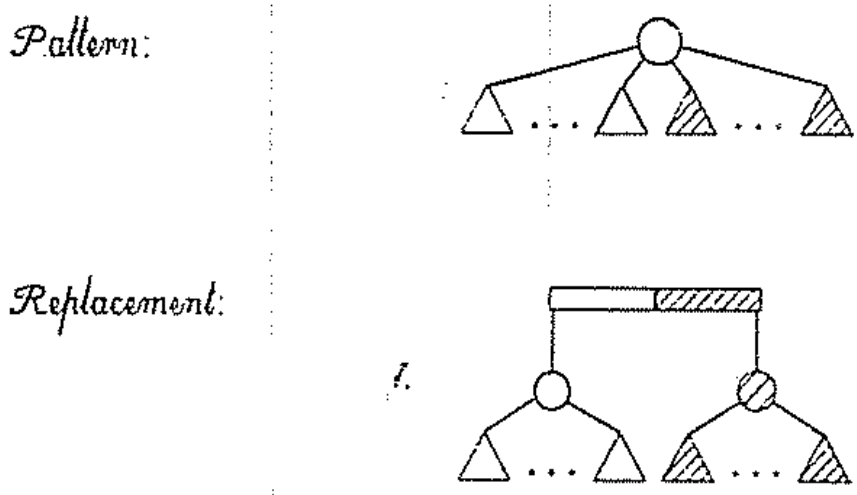


FIG. 9. Template $P3$ for a singly partial P-node which is not ROOT(T, S).

FIG. 10. Alternate replacements for Template $P3$.

In template P3 (above) we see that a new Q-node is created. In this case we will create a new R-node which consists of this new Q-node that was created. Since it used to be a P-node, there won't be an existing R-node to add this this new Q-node to.

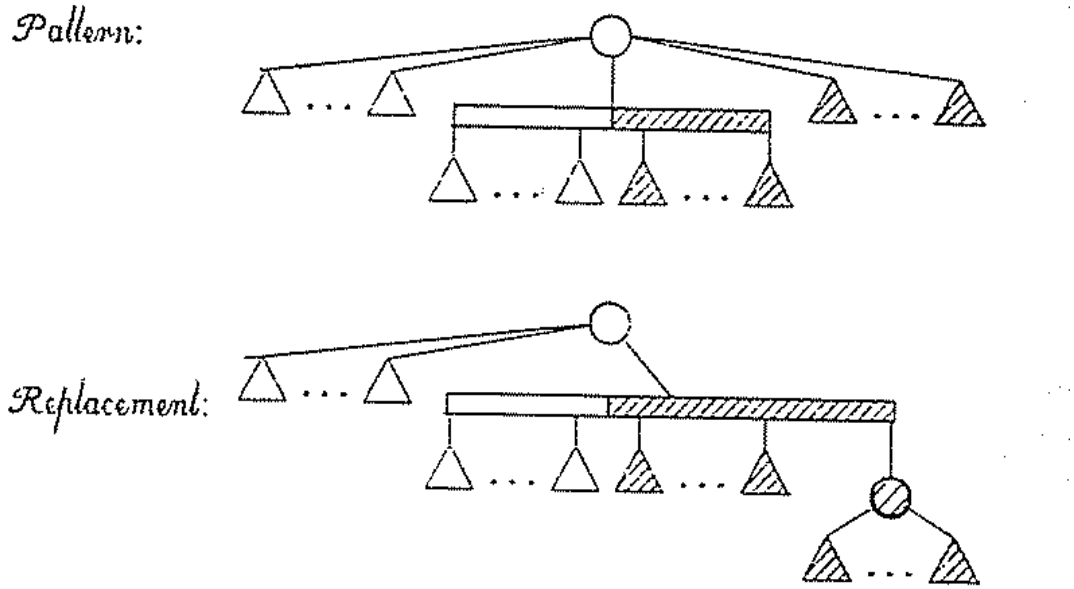


FIG. 11. Template $P4$ for ROOT(T, S) when it is a P-node with one partial child.

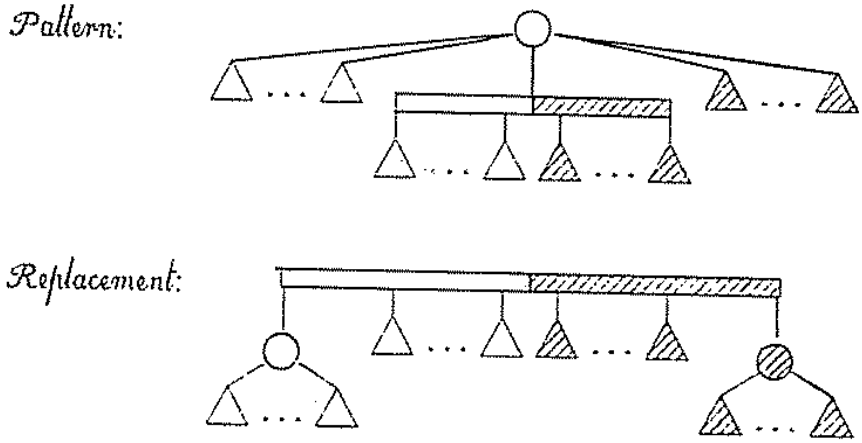


FIG. 12. Template $P5$ for a singly partial P-node, other than ROOT(T, S), with one partial child.

In templates P4 and P5 (above) we see that an existing Q-node is extended. In this case we will add this new Q-node to the R-node of the original Q-node. Then this new Q-node will be able to

be represented by an R-node.

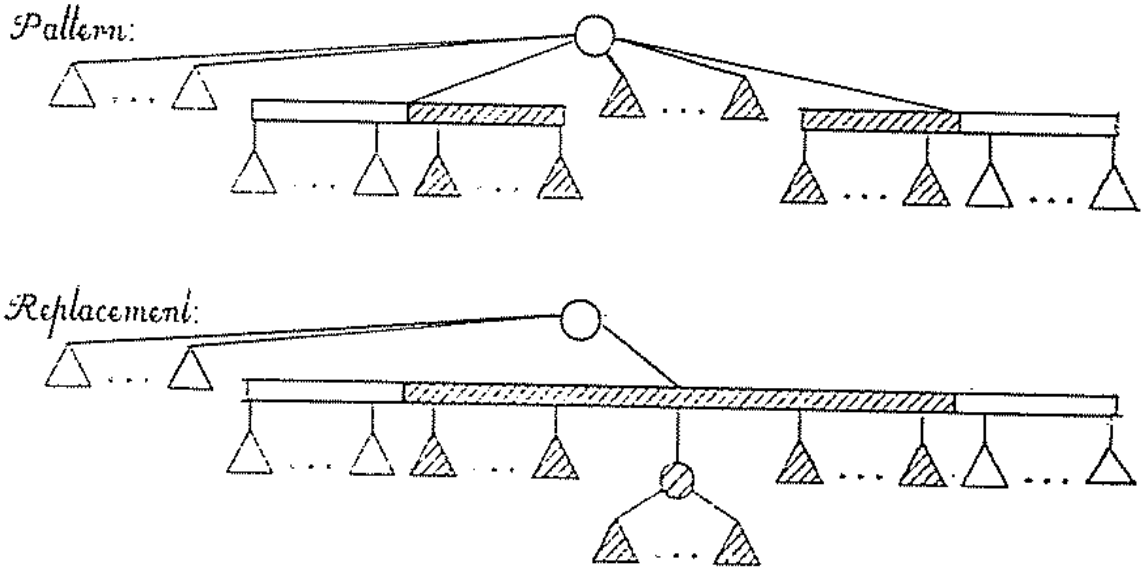


FIG. 13. Template *P6* for ROOT(T, S) when it is a doubly partial P-node.

In template P6 (above) we see that two existing Q-nodes are joined together. Let R_1 and R_2 be the R-nodes corresponding the Q-nodes respectively (note these will be different since these Q-nodes can not overlap by the definition of PQR-trees see **definition 3.1.1**). In this case we will add this new Q-node to each of R_1 and R_2 then we will union R_1 and R_2 to create R_3 which replaces both R_1 and R_2 (note: R3 will be able to represent all Q-nodes that both R_1 and R_2 represent and by the construction of R_1 and R_2 they will only overlap on this new Q-node meaning that R_3 will be a "tree").

We now provide the templates for Q-nodes. The templates Q0 and Q1 for Q-nodes are omitted since they are exactly the same as P1 except with a Q-node.

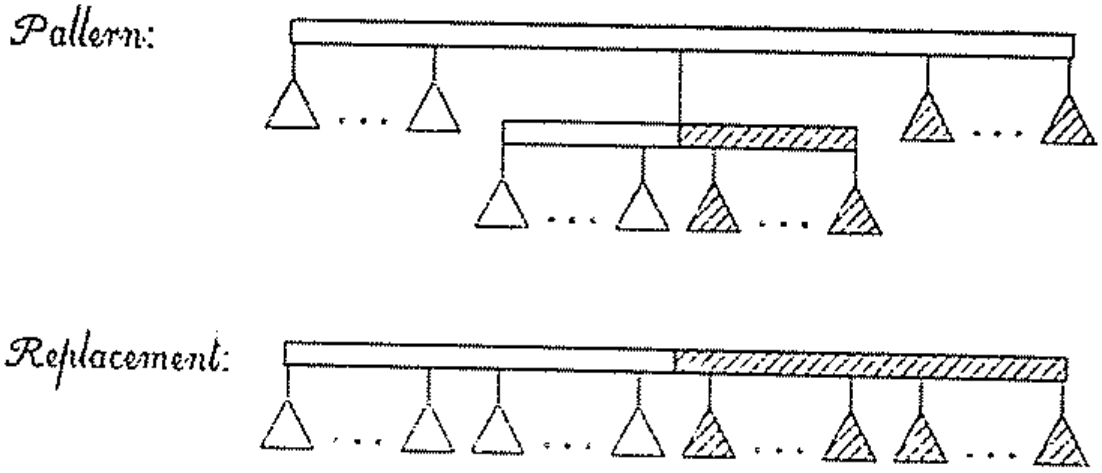


FIG. 14. Template $Q2$ for a singly partial Q-node.

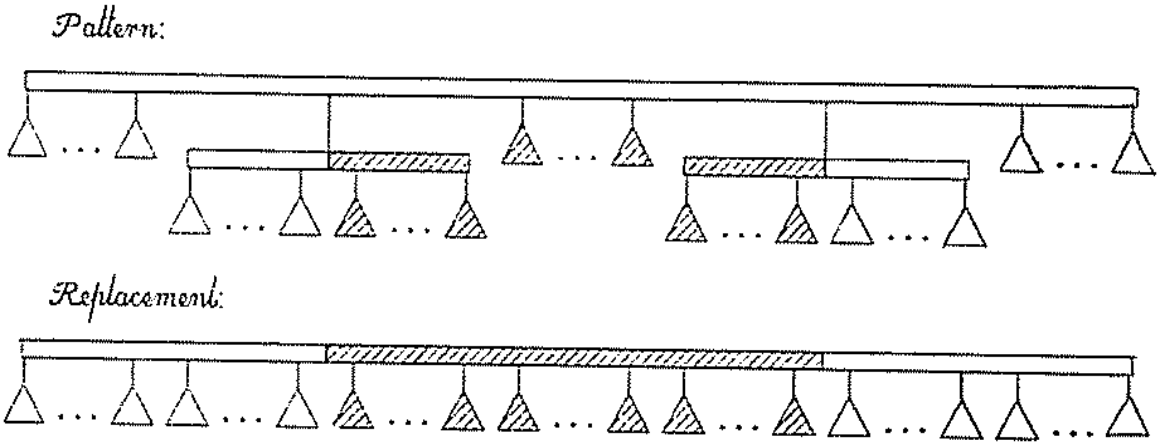


FIG. 15. Template $Q3$ for a doubly partial Q-node.

Similarly to P6 templates Q2 and Q3 (above) will cause R-nodes of the Q-nodes involved to be joined together and replaced by this new R-node, where the new Q-node was added to the original R-nodes.

In our algorithm (see **section 3.2.1**) we assume that when two PQ-trees T_1 and T_2 share a common subset of leaves L, then $T_1|_L = T_2|_L = X$ (i.e. the node X occurs in both T_1 and T_2). This means that any operations applied to X must "make sense" with respect to T_1 and T_2. Suppose without loss of generality that we are applying templates to X with respect to T_1.

We note that the leaf set of X must not change through the application of any of these templates.

This is because any change to the leaf set of X would change the leaf set of T_2. When X is a sub-Q-node, X will be stored with respect to an R-node. X will only change when applying templates Q2 or Q3 and it is clear from above that X will still be able to be properly represented by an R-node and that will be the only R-node representing X. When X is a P-node, it will either stay a P-node (in which case we do not need to worry about any changes with respect to T_2 as we will only be operating on the children of X) or it will become a sub-Q-node. In the latter case we will be creating a new R-node which will be unique for this Q-node and contain the Q-node as a sub-Q-node.