Intersection Graph Algorithms

Paul F. Dietz
Ph.D. Thesis

TR 84-628
August 1984

Department of Computer Science
Cornell University
Ithaca, New York 14853

INTERSECTION GRAPH ALGORITHMS

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Paul Frederick Dietz

August 1984

INTERSECTION GRAPH ALGORITHMS

Paul F. Dietz, Ph.D.
Cornell University 1984

An *intersection graph* for a set of sets C is a graph G together with a bijection from the vertices of G to C such that distinct vertices in G are adjacent if and only if their images under this bijection intersect. Of particular interest have been the classes of *chordal graphs*, the intersection graphs of sets of subtrees of a tree, and *interval graphs*, the intersection graphs of sets of intervals of the real line.

I examine another class of intersection graphs, the class of *directed path graphs*: intersection graphs of sets of paths in a directed tree. This class properly contains the class of interval graphs, and is properly contained by the class of chordal graphs. I give a linear time algorithm for recognizing directed path graphs and for constructing intersection representations, and a polynomial time algorithm for deciding directed path graph isomorphism.

Both algorithms use a data structure called a *partial path tree*, which is a kind of skeletal representation of the clique hypergraph of a directed path graph. I present linear time algorithms for finding partial path trees with specific roots and for finding partial path trees with arbitrary roots. I prove that partial path trees with identical roots are identical. Using this fact I develop a polynomial time algorithm for directed path graph isomorphism.

Biographical Sketch

Paul Dietz was born August 31, 1959 in Darby, Pennsylvania. He grew up in the suburbs of Baltimore, Maryland.

At the age of twelve, Paul took part in a research program developed by Doctor Julian Stanley of the Johns Hopkins University to identify mathematically and scientifically precocious youth. Paul ranked first in science and third in math among the participants.

With Dr. Stanley's encouragement, Paul entered Johns Hopkins University in the fall of 1974. He graduated three years later with a BES in electrical engineering.

Paul won an NSF graduate fellowship and enrolled in the Computer Science department of Cornell University in the fall of 1977. After spending four years at Cornell, he married Holly Cashatt, his childhood sweetheart. Paul went to the University of Southern California to teach while he finished his thesis.

Besides computer science, Paul is interested in speculative fiction and political philosophy.

To Holly

Acknowledgements

Many thanks to John Hopcroft for understanding and patience.

Thanks also to Merrick Furst for discussions that began the work described herein.

Thanks and apologies to all I kept waiting while finishing this thesis, especially Holly, and to all those who periodically prodded me when inertia threatened to overwhelm.

Finally, belated thanks to Dr. Stanley. Without his program my life would have been much more boring.

Table of Contents

List of Figures

1. Introduction

Many important problems in computer science can be formulated as graph problems. In their full generality, many of these problems are NP-complete and apparently intractable. Attention has therefore focused on specific classes of graphs on which the problems become tractable, such as trees, say, or planar graphs.

One useful way of producing classes of graphs is to consider graphs defined by the intersection of elements of some collection of sets. These so called *intersection graphs* arise in numerous applications, ranging from numerical analysis [39] and information retrieval [25] to molecular biology [2] [38].

Of particular interest is the class of *chordal graphs*, the class of graphs that are the intersection graphs of subtrees of a tree. Linear time algorithms exist for chordal graphs for the problems of finding maximal cliques, finding the chromatic number, finding maximum independent sets and minimum clique coverings.

A subclass of the chordal graphs is the class of *interval graphs*, the intersection graphs of collections of intervals on the real line. Fast algorithms exist for finding dominating sets in interval graphs and for determining interval graph isomorphism; these problems are NP-complete and isomorphism-complete (respectively) on chordal graphs.

Problems on chordal graphs can be rephrased as problems on clique hypergraphs.

The analogue of interval graph recognition is the *consecutive retrieval problem*: given a set of records and a collection of subsets of this set, find an ordering of the records in which all of these subsets occur consecutively. A linear time algorithm for this problem was discovered by Booth and Lueker [9]. The consecutive retrieval property has found widespread application; see [32].

This thesis examines a generalization of interval graphs called *directed path graphs*, the intersection graphs of directed paths in directed trees. The corresponding hypergraph problem is the *directed path tree problem*: given a set of records V and a collection of subsets of the records E, find a directed tree with vertices V in which every edge e in E induces a directed path.

This thesis presents a linear time algorithm for the directed path tree problem. The algorithm makes use of an intermediate structure called a *partial path tree*. Procedures are given that build a partial path tree with a specific root, that build a partial path tree with any root, and that convert a partial path tree to a directed path tree. All three algorithms run in linear time.

Partial path trees have the useful property that, for any hypergraph $H=(V,E)$ and any subset A of the vertices V, H has at most one partial path tree with root A. This property is exploited to give a polynomial (but unfortunately not linear) time algorithm for directed path graph isomorphism, solving an open problem of Booth and Johnson [8].

This thesis is arranged in six chapters. Chapter 1 is this introduction. Chapter 2

contains definitions from graph theory used in later chapters, definitions of intersection graphs and a survey of relevant results from the literature. Chapter 3 begins the discussion of the recognition and isomorphism algorithms for directed path graphs. A data structure, called a partial path tree, is defined. This data structure is a kind of skeletal representation of a directed path graph; having a partial path tree allows one to quickly find a directed path tree. Chapter 4 gives two algorithms for finding partial path trees, called *Find-PPT-With-Root* and *Find-PPT*. *Find-PPT-With-Root* decides if there is a partial path tree with a specified root. *Find-PPT* finds a root or concludes the hypergraph has no directed path tree. Both run in linear time. Chapter 4 also contains the linear time algorithm for finding a directed path tree from the partial path tree. Chapter 5 presents *DPHIso*, an algorithm that tests two directed path graphs for isomorphism. Chapter 6 concludes with a discussion of directions for further research.

2. Definitions and Background

We begin with a summary of definitions from basic graph theory. Notation and definitions are drawn from [4], [6] and [28].

2.1. Graphs

A *graph* is a pair of sets $G=(V,E)$, where V is a set of *vertices*, and E is a set of unordered pairs of distinct vertices called *edges*. If G is a graph, $V(G)$ and $E(G)$ are the vertex and edge sets of G, respectively. A vertex u is *adjacent* to a vertex v if $\{u,v\}$ is an edge, i.e., $\{u,v\} \in E$.

The set of vertices adjacent to u is $Adj_G(u)$. An edge $e=\{u,v\}$ is *incident with* the vertices u and v, which are the *ends* of e. The set of edges in graph G incident with vertex v is $Inc_G(v)$. The *degree* of v, $d(v)$, is $|Inc(v)|$ (in general, subscripts will be omitted when they are clear in context).

Two graphs G and H are *isomorphic* if there is a bijection ϕ from $V(G)$ to $V(H)$ such that vertices u and v of G are adjacent in G iff $\phi(u)$ and $\phi(v)$ are adjacent in H. The function ϕ is an *isomorphism* from G to H. An isomorphism from G to itself is an *auto*morphism. *A labelled isomorphism* is an isomorphism that also preserves some labelling function of the vertices or the edges of the graphs.

A graph H is a *subgraph* of a graph G iff $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. If $E(H)=\{e \in E(G) : e \subseteq V(H)\}$ then H is the subgraph of G induced by $V(H)$, denoted $G[V]$ (where V is the vertex set of H). H is called a *vertex induced subgraph of* G. If $V(H) = \{v \in V(G) : v \in e$ for some $e \in E(H)\}$ then H is the

subgraph of G induced by $E(H)$, denoted $H[E]$ (where E is the edge set of H). H is called an *edge induced subgraph of* G. If $V \subseteq V(G)$ ($E \subseteq E(G)$) then let $G\text{-}V = G[V(G)\text{-}V]$ ($G\text{-}E = G[E(G)\text{-}E]$) be the graphs obtained from G by deleting the vertices in V (the edges in E).

Let G be a graph. A *walk* is a sequence $P = v_0,e_1, \ldots ,v_{k-1},e_k,v_k$ of vertices v_i and edges e_j such that $e_i=\{v_{i-1},v_i\}$, $1 \leq i \leq k$. A *tour* is a walk in which all edges are distinct. A *path* is a tour in which all vertices are distinct. The length of the walk (tour, path) P given above is k. The first vertex (v_0) and last vertex (v_k) in a path are called the *ends* of the path; a path with ends u and v is called a (u,v) *path*.

Two vertices u and v in G are *connected* if G has a (u,v) path. A graph is connected if all pairs of vertices are connected. A *connected component* of G is a maximal connected subgraph of G.

A *cycle* is a tour of length k, $k > 2$, where $v_0=v_k$ and $v_0, \ldots ,v_{k-1}$ are distinct. A *tree* is a connected graph with no cycles.

A *clique* of a graph G is a vertex induced subgraph $C=G[V(C)]$ such that for each vertex v in $V(C)$, $Adj_C(v) = V(C)\text{-}\{v\}$.

If $C = v_0,e_1, \ldots ,e_{k-1},v_k$ is a cycle in G, a *chord* of C is an edge e in $E(G)$ connecting vertices v_i and v_j such that $e \neq e_i$ for any $i=1, \ldots ,k$. A graph is *chordal* if every cycle containing at least four vertices has a chord.

2.2. Directed Graphs

A *directed graph*, or *digraph*, is a pair of sets (V,A), where V is a set of vertices and A is a set of ordered pairs of distinct vertices, called *arcs*. If D is a digraph then $V(D)$ and $A(D)$ are the vertex and arc sets of D, respectively. An arc $a=(u,v)$ is incident with u and v; the vertices u and v are the *head* and *tail* of a; a is an *in-arc* of v and an *out-arc* of u. The sets of out-arcs and in-arcs of vertex u are $Inc^+(u)$ and $Inc^-(u)$, respectively. The in-degree of u is $d^-(u)=|Inc^-(u)|$, the out-degree $d^+(u)=|Inc^+(u)|$. The sets of tails of arcs in $Inc^+(u)$ and heads of arcs in $Inc^-(u)$ are $Adj^+(u)$ and $Adj^-(u)$, respectively.

Two digraphs D and D' are *isomorphic* if there is a bijection ϕ from $V(D)$ to $V(D')$ such that arc (u,v) is in $E(D)$ iff arc $(\phi(u),\phi(v))$ is in $E(D')$. As in graphs, an isomorphism from D to itself is an *auto*morphism.

A *directed walk* is a sequence $P = v_0,a_1, \ldots ,v_{k-1},a_k,v_k$ of vertices and arcs such that $a_i=(v_{i-1},v_i)$. A *directed tour* is a directed walk in which all arcs are distinct. A *directed path* is a directed tour in which all vertices are distinct. A directed path with first vertex u and final vertex v is a *(u,v)-directed path*. As with paths in graphs, the length of a directed path is the number of arcs in the path.

A *directed cycle* is a directed tour of length $k > 1$ in which $v_0=v_k$ and vertices $v_0, \ldots ,v_{k-1}$ are distinct.

A *directed acyclic graph*, or *DAG*, is a digraph D with no cycles. Let u,v be vertices in $V(D)$. The vertex u is an *ancestor* of v (and v a *descendant* of u) if

there is a (u,v)-directed path in D; otherwise, u and v are *unrelated*. If u is an ancestor of v and (u,v) is an arc then u is a *parent* of v, v a *child* of u.

A *rooted tree* (or *directed tree*) is a directed acyclic graph in which all vertices have indegree 1 save one, the root, which has indegree 0. The root of a rooted tree T is denoted $root(T)$. The *subtree of T rooted at v* is the subtree of T induced by the descendants of v. The *depth* of a vertex v in a rooted tree T, denoted $depth(v)$, is the length of the path from $root(T)$ to v. The depth of a rooted tree is $depth(T) = max\{depth(v) : v \in V(T)\}$. A *leaf* is vertex in a DAG with no children. An *ordered tree* is a rooted tree in which a linear ordering is given for every nonleaf's children. The *frontier* of vertex v in an ordered tree, denoted $FRONTIER(v)$, is the sequence of leaves that are descendants of v, where a leaf a occurs before leaf b in the sequence iff there is a common ancestor c of a and b such that the child of c on the path from c to a comes before the child of c on the path from c to b. In other words, the frontier of vertex v is the sequence of leaves visited by a traversal of the subtree rooted at v. If T is an ordered tree then $FRONTIER(T) = FRONTIER(root(T))$.

2.3. Families

A *family* is an indexed collection of (not necessarily distinct) elements $F = \{F(i) : i \in I(F)\}$. An element x is in a family F (x an element of F) iff there is an i in $I(F)$, $x = F(i)$. A family F' is a *subfamily* of a family F (and F a *superfamily* of F'), written $F' \subseteq F$, if $I(F') \subseteq I(F)$ and, for every i in $I(F')$, $F'(i) = F(i)$. The intersection of two families is the maximal family that is a

subfamily of both; the union of two families is the minimal family that is a superfamily of both (if any such family exists). If F' is a subfamily of F then the difference F-F' is the subfamily of F with $I(F\text{-}F') = I(F)\text{-}I(F')$.

A bijection from a family F to a family F' is a bijection from $I(F)$ to $I(F')$. If e is an element of family F ($e = F(i)$, i an element of $I(F)$) then the image of e under bijection θ is denoted $\theta(e)$ ($=F'[\theta(i)]$). Two families F and F' are equal if there is a bijection θ from F to F' such that $\theta(e)=e$ for all e in F.

The *size* of a family of sets F is $SIZE(F) = \sum_{i \in I(F)} |F(i)|$.

2.4. Hypergraphs

A *hypergraph* is a pair (V,E), V a set of *vertices* and E a family of sets of vertices called *edges*.

Two hypergraphs H and H' are *isomorphic* iff there is a pair (ϕ,θ) of bijections from $V(H)$ to $V(H')$ and $E(H)$ to $E(H')$ such that for any vertex v in $V(H)$ and edge e in $E(H)$, v is an element of e iff $\phi(v)$ is an element of $\theta(e)$. The pair (ϕ,θ) is an isomorphism from H to H'.

A hypergraph H' is a *subhypergraph* of a hypergraph H if $V(H')$ is a subset of $V(H)$ and $E(H')$ is a subset of $E(H)$. Subhypergraphs, vertex induced subhypergraphs, edge induced subhypergraphs, adjacency and incidence are defined as for graphs.

A *hyperwalk* is a sequence $P = v_0,e_1, \ldots ,v_{k-1},e_k,v_k$ of vertices and edges such

that the vertices v_{i-1} and v_i are elements of the edge e_i, $1 \leq i \leq k$. As in walks and directed walks, the length of this hyperwalk P is k. The *ends* of the hyperwalk P are the vertices v_0 and v_k. A *hypertour* is a hyperwalk in which all the edges are distinct. A *hyperpath* is a hypertour in which all edges and vertices are distinct. A *simple hyperpath* is a hyperpath in which

$$e_i \cap e_j = \begin{cases} \{v_i\} & \text{if } i = j\text{-}1 \\ \emptyset & \text{if } i < j\text{-}1 \end{cases} \tag{1}$$

where $1 \leq i < j \leq k$. A hyperwalk (hypertour, hyperpath, simple hyperpath) with ends u and v is a (u,v) *hyperwalk (hypertour, hyperpath, simple hyperpath).*

Two vertices u and v are *connected* in a hypergraph if the hypergraph has a (u,v) hyperpath. A hypergraph is connected if every pair of vertices are connected. A *connected component* of a hypergraph is a maximal connected subhypergraph.

A *hypercycle* is a hypertour in which $v_0, \ldots, v_{k-1}$ are distinct. A *simple hypercycle* is a hypercycle for which equation (1) holds.

The *dual* of a hypergraph $H=(V,F)$ is the hypergraph $H^+=(I(F),F')$, where F' is the family $F' = \{Inc_H(v) : v \in V\}$.

The *clique hypergraph* of a graph G is a hypergraph $C(G)$ with vertices $V(G)$ and edges the maximal cliques of G.

The edges of a hypergraph satisfy the *Helly property* if for any subset $A \subseteq E(H)$, if for all edges e and e' in A the intersection of e and e' is nonempty, then $\cap A$ is nonempty.

If H is a hypergraph and V' is a set of vertices, $V(H)$ and V' disjoint, $H+V'$ is the hypergraph $(V(H) \cup V', E(H))$. Similarly, if E' is a family of subsets of $V(H)$, $I(E(H))$ and $I(E')$ disjoint, let $H+E'$ be the hypergraph $(V(H), E(H) \cup E')$. If e is an edge in $E(H)$ then let H_e be the hypergraph obtained from H by:

1. adding a new vertex v_e to $V(H)$,
2. replacing every edge f in $E(H)$ that intersects e with $f\text{-}e \cup \{v_e\}$, and
3. deleting the vertices in e from $V(H)$.

This process is called *contracting an edge.* A similar process may be defined for graphs and directed graphs, except that any edge or arc with both ends in the set e is deleted.

2.5. Intersection Graphs

A family of sets F is an *intersection representation* of a graph G (and G the *intersection graph* of the family F) if $I(F)=V(G)$ and for any distinct vertices x and y in $V(G)$, $F(x)$ intersects $F(y)$ iff x and y are adjacent in G.

An *interval graph* is the intersection graph of a family of intervals of the real line. A *proper interval graph* is an intersection graph of a family of intervals on the real line, such that no no interval is a subset of any other. A *path graph* is the intersection graph of a family of paths in a tree. A *directed path graph* is the intersection graph of a family of directed paths in a rooted tree. A *circular-arc graph* is the intersection graph of a family of arcs of a circle. A *proper circular-arc graph* if it is the intersection graph of a family of arcs of a circle, no arc in the family a subset of another.

Gavril and others [21] [11] have proved the following fundamental theorem:

Theorem 2-1:

Let G be a graph. The following three statements are equivalent:

1. G is chordal.
2. G is the intersection graph of a family of subtrees of a tree.
3. There is a tree T whose vertices are the maximal cliques of G such that, for every vertex v in G, the set of maximal cliques in G that contain v induces a connected subgraph of T. (The tree T is called a *characteristic tree* for G.)

Proof:

See [28], pages 92-93. #

Similar theorems have been proved for interval graphs, path graphs and directed path graphs.

Theorem 2-2:

G is an interval graph iff the maximal cliques of G can be linearly ordered such that for any x in $V(G)$ the cliques containing vertex x are consecutive in the linear order.

Proof:

See [26], [28], pages 172-173. #

Theorem 2-3:

G is a directed path graph iff there is a directed tree T whose vertices are the maximal cliques of G such that, for any vertex x in $V(G)$, the set of maximal cliques containing x induce a directed path in T

Proof:

See [22], theorem 2.1. #

Theorem 2-4:

G is a path graph iff there is a tree T whose vertices are the maximal cliques of G so that for any vertex x in $V(G)$, the set of maximal cliques containing x induce a path in T.

Proof:

See [23], theorem 3.3. #

The sequence of cliques in theorem 2-2 and the trees in theorems 2-3 and 2-4 are called *characteristic trees* (for interval graphs, directed path graphs, or path graphs). An immediate consequence of these theorems is:

Corollary 2-5:

Proper interval graphs are interval graphs, which are directed path graphs, which are undirected path graphs, which are chordal graphs.

2.6. Algorithms on Intersection Graphs

2.6.1. Chordal Graphs

Rose, Tarjan and Lueker have [40] an elegant algorithm for recognizing chordal graphs. Their algorithm exploits the following theorem. A vertex v in $V(G)$ is *simplicial* if $G[Adj(v)]$ is a clique. A *perfect elimination scheme* is a linear ordering $v_1, \ldots, v_n$ of the vertices of G such that v_i is a simplicial vertex in $G\text{-}\{v_1, \ldots, v_{i-1}\}$, $1 \leq i \leq n$.

Theorem 2-6:

A graph G is chordal iff it has a perfect elimination scheme.

Proof:

See [28], pages 83-84. #

Rose, Tarjan and Lueker's algorithm constructs a perfect elimination scheme, if one exists. The algorithm is straightforward, and is given below (figure 2-1). The algorithm uses a technique called *lexicographic breadth-first search*. It can be implemented to run in linear ($O(|V|+|E|)$) time. A simpler algorithm, attributed to Tarjan, that also produces perfect elimination schemes in chordal graphs is called *maximum cardinality search* [28]. It uses a technique similar to lexicographic breadth-first search, except that the vertex picked in each iteration is the unnumbered vertex that is adjacent to the most numbered vertices.

To test if (V,E) is chordal we must also be able to test if a linear ordering of the

Comment Rose, Tarjan and Lueker algorithm for finding perfect elimination schemes in a chordal graph (V,E);

proc *RTL(V,E)*

1. **for all** v in V **do** $LABEL[v]$, $\sigma(v) := \emptyset$, 0;
2. **for** $i := |V|$ **to** 1 **step** -1 **do**
3. Pick a vertex v from V such that $\sigma(v)$=0 and $LABEL[v]$ is maximum;
4. $\sigma(v) := i$;
5. **for all** u in $Adj(v)$ **do** $LABEL[u] := LABEL[u] \cup \{i\}$

end

6. **return** σ

end *RTL*

Figure 2-1: Lexicographic Breadth-First Search

vertices σ is, in fact, a perfect elimination sequence. This can also be done in linear time (figure 2-2).

(The algorithms in figures 2-1 and 2-2 are drawn from [28], chapter 4. Implementation details and proofs of correctness may also be found there.)

Let $\sigma(1), \ldots, \sigma(|V|)$ be a perfect elimination scheme on a chordal graph G, and let C be a maximal clique in G. If the integer $m = min\{\sigma^{-1}(v) : v \in V(C)\}$ and the graph $G_i = G\text{-}\{v_1, \ldots, v_{i-1}\}$ then $V(C)$ is simply $\{v_m\} \cup Adj_{G_m}(v_m)$. Therefore,

Comment Test if σ is a perfect elimination scheme on (V,E);

proc *Perfect*(σ,V,E)

1. **for all** v in V **do** $A[v] := \emptyset$;
2. **for** $i := 1$ **to** n-1 **do**
3. $\quad v := \sigma(i)$;
4. $\quad X := \{x \in Adj(v) : \sigma^{-1}(x) < \sigma^{-1}(v)\}$;
5. $\quad$ **if** $X \neq \emptyset$ **then**
6. $\quad\quad u := \sigma(min\{\sigma^{-1}(x) : x \in X\})$;
7. $\quad\quad A[u] := A[u] \cup (X\text{-}\{u\})$;
8. $\quad$ **if** $A[v]$ is not a subset of $Adj(v)$ **then**
9. $\quad\quad$ **return false**;
10. **return true**

end *Perfect*

Figure 2-2: Testing Perfect Elimination Schemes

Proposition 2-7:

[18] A chordal graph (V,E) has at most $|V|$ maximal cliques, and has exactly $|V|$ maximal cliques iff E is empty.

In fact, there exists an algorithm for finding the maximum cliques of (V,E) given a perfect elimination scheme. The algorithm runs in linear time (so we know the sum of the sizes of the cliques is $O(|V|+|E|)$). Linear time algorithms also exist for

minimum colorings, maximum independent sets and minimum clique covers of chordal graphs [20]. All of these problems (and the problem of finding a maximum clique) are NP-complete on general graphs [19].

On the other hand, Booth and Johnson [8] has shown that the problem of finding a minimum dominating set on chordal graphs is NP-complete, and Lueker and Booth [35] have shown that testing isomorphism of chordal graphs is polynomially equivalent to general graph isomorphism (see [7] for a list of isomorphism-complete problems).

Of interest to us is the fact that all maximum cliques of a chordal graphs can be found in linear time. In other words, given a chordal graph G we can find the clique hypergraph $\mathcal{C}(G)$ in linear time. This will simplify the problem of recognizing interval, directed path and path graphs, since a chordal graph G is an interval (direct path, path) graph iff we can arrange the vertices in the dual of the clique hypergraph, $\mathcal{C}(G)^+$ into a path (directed tree, tree) in which all edges in $\mathcal{C}(G)^+$ induce (directed) paths.

Proposition 2-8:

The dual of the clique hypergraph of a chordal graph may be found in $O(|V|+|E|)$ time.

Let H be a hypergraph. We call the problem of deciding of there is a tree T such that the edges in $E(H)$ induce subtrees in T the *subtree representation problem*; a hypergraph for which such a tree exists is called a *chordal hypergraph*.

Proposition 2-9:

If G is chordal then $C(G)^+$ is a chordal hypergraph.

In addition,

Theorem 2-10:

If H is a chordal hypergraph then the edges of H satisfy the Helly property.

Proof:

Let T be a tree with vertices $V(H)$ in which each edge in $E(H)$ induces a subtree. Let $e_1, \ldots, e_k$ be edges in $E(H)$, $e_i \cap e_j \neq \emptyset$, $1 \leq i,j \leq k$. Let U_i be the subgraph of T induced by $e_1 \cap \ldots \cap e_i$, $1 \leq i \leq k$. We prove that each U_i is nonempty. Clearly, U_1 and U_2 are nonempty. Inductively, if U_{i-1} is nonempty ($i < 2$), and if $V(T_i)$ does not intersect $V(U_{i-1})$ then T_i is a subtree of one of the connected components in $T\text{-}V(U_{i-1})$. But none of these components contains vertices from all of $e_1, \ldots, e_{i-1}$, so $e_i \cap e_j$ is empty for some j, $1 \leq j < i$. This is a contradiction, so $V(T_i)$ and $V(U_{i-1})$ intersect, and U_i is nonempty. #

2.6.2 Interval Graphs and the Consecutive Retrieval Problem

Let H be a hypergraph. The *consecutive retrieval problem* on H is the problem of finding a linear ordering in which, for every edge in $E(H)$, the vertices in the edge occur consecutively. We say H is *CRP* if there exists such an ordering.

By theorem 2-2, a graph is an interval graph iff the dual of its clique hypergraph is CRP. Booth and Lueker gave a linear time algorithm for determining if a hypergraph is CRP [9] [10] [36]. Their algorithm uses an important data structure called a *PQ tree*. The algorithms described later for recognizing directed path graphs make essential use of a variant of PQ trees called *PQR trees*.

A PQ tree represents a set of linear orderings of some set L. A PQ tree is a rooted, oriented tree that has three kinds of vertices:

1. Leaves. The leaves of a PQ tree are the elements of the set L (except for the empty PQ tree, which has no nodes and represents the empty set of sequences).

2. P nodes. These internal nodes have at least two children and are represented by circles.

3. Q nodes. These internal nodes have at least two children and are represented by rectangles.

See figure 2-3.

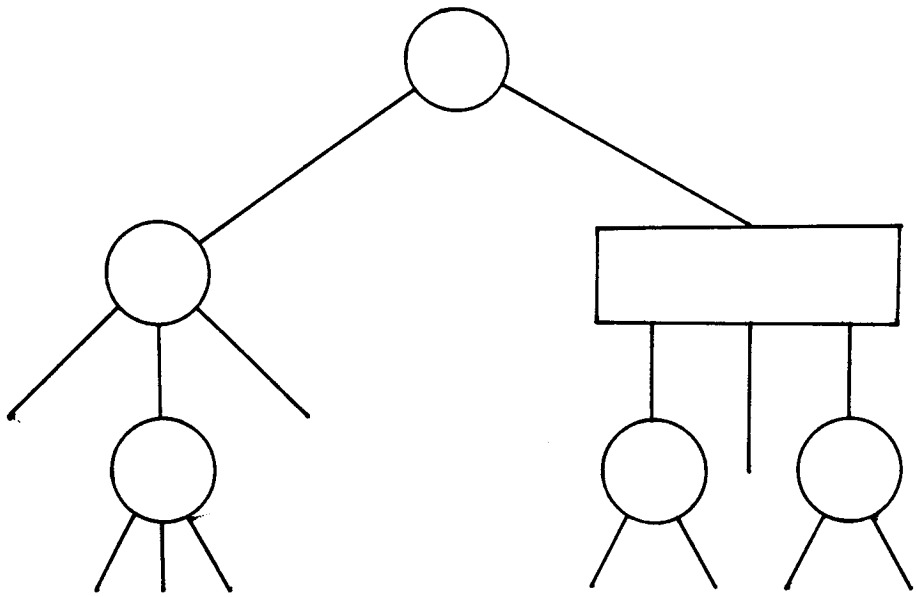

Figure 2-3: A PQ Tree

Two PQ trees T and T' are *equivalent* (denoted $T \equiv T'$) if there is a sequence comprised of the following transformation that turns T into T':

1. Arbitrarily permute the order of the children of a P node.
2. Reverse the order of the children of a Q node.

Clearly, $\equiv$ is an equivalence relation. Define

$$CONSISTENT(T) = \{FRONTIER(T') : T \equiv T'\}$$

to be the set of sequences represented by the PQ tree T, where $CONSISTENT(T) = \emptyset$ if T has no vertices.

One can show that

Theorem 2-11:

If Z and Z' are PQ trees, and

$$CONSISTENT(Z) = CONSISTENT(Z')$$

then Z and Z' are equivalent.

Proof:

See [35], theorem 1. #

Booth and Lueker have a procedure, *REDUCE* , such that if T is a PQ tree with leaves elements of a set L and A is a nonempty subset of L, then

$$CONSISTENT(REDUCE(T,A)) =$$

$$\{\sigma \in CONSISTENT(T) : A \text{ is consecutive in } \sigma\}.$$

They also gave an implementation of PQ trees and *REDUCE* so that n calls to *REDUCE* on sets $A_1, \ldots, A_n$ can be performed in $O(|L|+\sum_{i=1}^{n} |A_i|)$ time. This solves the consecutive retrieval problem, and, when combined with the algorithm for extracting maximum cliques from a chordal graph, allows one to recognize and construct intersection representations for interval graphs in $O(|V|+|E|)$ time.

Booth and Lueker use PQ trees to help implement the Lempel-Even-Cederbaum planarity testing algorithm ([9], [16], section 8.4), achieving linear running time. A generalization of PQ trees, called PQ graphs, has been used to get a (nearly)

linear time algorithm for the graph realization problem (the problem of deciding if a (0,1) matrix is a fundamental circuit matrix of a graph [17]). Lueker and Booth [35] and Colbourn and Booth [13] have used PQ trees in linear time algorithms to decide isomorphism of interval graphs, to find canonical forms for (labelled) PQ trees, to find the number of automorphisms of an interval graph, and to find the generators of the automorphism group of interval graphs. The algorithms are extensions of the well known linear time tree isomorphism algorithm [1], [12]. This algorithm will be described later in chapter 5.

Bertossi [5] has a fast algorithm for finding hamiltonian circuits in proper interval graphs. The complexity of finding hamiltonian circuits in more general classes of chordal graphs is unknown.

2.6.3. Path Graphs and Directed Path Graphs

The problems of recognizing path graphs and directed path graphs can be reduced to two problem on hypergraphs: the path tree problem (PTP): given a hypergraph H, find a tree with vertices $V(H)$ in which all edges in $E(H)$ induce paths, and the directed path tree problem (DPTP): given a hypergraph H find a directed tree with vertices $V(H)$ in which all edges in $E(H)$ induce directed paths?

Gavril [22] [23] has $O(|V|^4)$ time algorithms for these problems.

Truszczynski [44] has a worse than linear time algorithm for the directed path tree problem. His algorithm used a data structure similar to the PQR trees that are used here. His paper described an application of directed path tree problem to

information retrieval: suppose a given set of records is to be stored on a disk. Let each record have one pointer that can point to one other record. Require the resulting directed graph to be acyclic. Given a collection Q of subsets of the records, arrange the pointers so that the records in each subset occur consecutively. The rational for this problem is that each "query" in Q can be represented by its length, and by a pointer to its first record in the tree. Truszczynski also has a polynomial time algorithm for the problem where the requirement that the graph be acyclic has been dropped [42].

Booth and Johnson [8] have a polynomial time algorithm for finding minimum dominating sets in directed path graphs. Their algorithm runs in linear time if a characteristic tree for the directed path graph is available; the algorithm presented later in chapter 4 produces a characteristic tree in linear time. Lueker and Booth's [35] proof that chordal graph isomorphism is isomorphism-complete extends naturally to show that path graph isomorphism is also isomorphism-complete (the chordal graphs constructed in the reduction are actually undirected path graphs). Booth and Johnson conjecture that isomorphism of directed path graphs can be done in polynomial time (by analogy with the minimum dominating set problem). Their conjecture is true; an algorithm appears in chapter 5.

3. Directed Path Hypergraphs and Partial Path Trees

In this chapter we investigate the structure of directed path hypergraphs.

In the first section (section 3.1) we prove a very (useful) technical lemma, lemma 3-1, that will allow us to prove many particular hypergraphs are not directed path hypergraphs. The lemma allows one to reduce a complicated hypergraph H to a simpler hypergraph H' so that if H has a directed path tree then so does H'. We will use this lemma along with lemma 3-2 (which gives some simple hypergraphs without directed path trees) to prove that the algorithms in section 4 are correct when they report a hypergraph has no directed path tree.

In section 3.2 we define *partial path trees*. The vertices of a partial path tree form a partition of the vertex set of the hypergraph; these trees can be thought of as decompositions of directed path hypergraphs into interval hypergraphs. Partial path trees are used in both the recognition and isomorphism algorithms. In the recognition algorithm we solve each of the interval hypergraphs using PQ tree techniques, then knit the solutions together. In the isomorphism algorithm, we exploit theorem 3-5 which states that PPT with identical roots are identical, and lemma 3-6, which states that if a hypergraph H has a directed path tree with root r it has a partial path tree with root $\{r\}$.

We next define the sets $m(x)$ and $M(x)$ (where x is a vertex in some PPT P for hypergraph H), and prove lemma 3-7, which shows (among other things) that both

of these sets induce paths in any directed path tree for H. Using $M(x)$ we then prove lemma 3-8, which shows that certain other sets also induce directed paths in directed path tree for H. These sets, which are in the *connector* of a vertex x (in P) with its parent will be used later to fit together the PQ trees for the vertices of P to get a directed path tree for H. Lemma 3-9 shows how these connectors must be arranged in any directed path graph for H.

Lemma 3-10 shows that if H has a directed path tree then any PPT P for H is also a PPT for the *closure* of H, obtained by repeatedly intersecting and joining edges in H (operations 2 and 5; see below). This lemma is used in the proof of correctness of the first algorithm in chapter 4 (lemma 4-1), and in the proof of correctness of the second algorithm (lemma 4-10).

Finally, lemma 3-11 shows that if vertex u is an ancestor of vertex v in a PPT P for H, one can find a simple hyperpath in the closure of H from some vertex in u to some vertex in v, the edges in the hyperpath intersecting only ancestors of v. As a consequence, if T is a subtree of P with the same root then for any directed path tree D for H, the subgraph of D induced by the union of the vertices in T is a subtree of D.

3.1. Simplification and Structural Theorems

Let H be a hypergraph, and let T be a rooted tree with vertices $V(H)$. Consider the following six operations:

1. (Contract an edge). Set the hypergraph H to be H_e and set the tree T to be T_e (see the definition of edge contraction), where e is an edge in $E(H)$.

2. (Intersect two edges). Set the hypergraph H to be $H+\{c \cap d\}$, . Leave the tree T unchanged. The edges c and d in $E(H)$ and the set $c \cap d$ must be nonempty.
3. (Delete a vertex). Set the hypergraph H to be $H-\{r\}$ for some vertex r in $V(H)$. If r is the root of the tree T, one of its children becomes the new root and inherits r's other children. Otherwise, r's children are inherited by r's parent.
4. (Delete an edge). Set the hypergraph H to be $H-\{e\}$ for some edge e in $E(H)$. Leave the tree T unchanged.
5. (Join two edges). Set the hypergraph H to be $H+\{c \cup d\}$. The tree T is left unchanged. The sets c and d must be edges in $E(H)$ and there must exist an edge e in $E(H)$ such that c and d are subsets of e, and $c \cap d$ is nonempty.
6. (Difference of two edges). Set the hypergraph H to be $H+\{c\text{-}d,d\text{-}c\}$, the tree T is unchanged. The edges c and d in $E(H)$ must intersect, neither c nor d can be a subset of the other and both must be subsets of some edge e in $E(H)$.

If hypergraph H' and tree T' are obtained from hypergraph H and tree T by a sequence of such operations then the pair (H',T') is a *simplification* of the pair (H,T), and this sequence of operations *simplifies* the pair (H,T) to the pair (H',T') (and simplifies H to H').

Lemma 3-1:

If T is a directed path tree for hypergraph H and the pair (H,T) simplifies to the pair (H',T') then T' is a directed path tree for the hypergraph H'.

Proof:

The proof of the lemma is by induction on the number of operations, so we need only show what happens in each of the six cases.

1. $H' := H_e$, $T' := T_e$. If an edge f in $E(H)$ is disjoint from edge e then $T'[f]=T[f]$, so f induces a directed path in T'. Otherwise, if an edge f intersects the edge e then $T[f]$ can be decomposed into at most three directed paths: a directed path P_1 that contains proper ancestors of vertices in e, a directed path P_2 that contains only vertices in e, and a directed path P_3 that contains proper descendants of vertices in e, where $V(P_1)$ and $V(P_3)$ are disjoint from e, the tail of P_1 is the head of P_2, the tail of P_2 is the head of P_3, and $V(P_2)$ is not empty. Since P_1 and P_3 share no arcs with $T[e]$, $P_1\text{-}e \cup \{v_e\}$ and $P_3\text{-}e \cup \{v_e\}$ are directed paths in T', and their union is a directed path with vertices $f\text{-}e \cup \{v_e\}$. Therefore, T' is a directed path tree for H'.
2. $H' := H+\{c \cap d\}$, $T' := T$. Since the intersection of two directed paths of directed tree is a directed path, T' (which equals T) is a directed path tree for H'.

3. H'=H-$\{r\}$. As in case 1, for any edge f in $E(H)$ containing vertex r, f can be broken down into three directed paths consisting of: proper ancestors of r in T, proper descendants of r, and r itself. In T', the vertex r has been deleted and the other two lists are linked together, so f-$\{r\}$ is a directed path in T'.
4. $H' := H$-$\{e\}$, $T' := T$. All remaining edges in $E(H)$ are still directed paths in T' (=T), so T' is a directed path tree for the hypergraph H'.
5. $H' := H$+$\{c \cap d\}$, $T' := T$, where edge c intersects edge d and c and d are subsets of some edge e. Since c and d are subsets of e, they must both induce subpaths in T of the directed path $T[e]$. Since they intersect, these paths must overlap, so their union must induce a directed path. Therefore, T' (=T) is a directed path tree for H'.
6. $H' := H$+$\{c$-d, d-$c\}$, $T' := T$, where edges c,d,e are as in the previous case, and neither c nor d is a subset of the other. As in case 5, c and d are subpaths of $T[e]$. Since neither is a subset of the other, it cannot be the case that there are vertices in c-d that are descendants of all vertices in d and vertices in c-d that are ancestors of all in d. Therefore, $c \cup d$ can be decomposed into three nonempty subpaths $T[c$-$d]$, $T[c \cap d]$ and $T[d$-$c]$, where the second occur between the other two. Therefore, c-d and d-c induce directed paths in T' (=T), so T' is a directed path tree for H'. #

To prove that a particular hypergraph H is not a directed path hypergraph it suffices to simplify it to a hypergraph having no directed path tree. The next lemma provides examples of such hypergraphs.

Lemma 3-2:

The following hypergraphs have no directed path trees:

HC_i: $(\{v_1, \cdots, v_i\},$
$\{\{v_1,v_2\}, \cdots, \{v_{i-1},v_i\},\{v_i,v_1\}\})$, $i > 2$

I_2: $(\{a,b,c,d\},$
$\{\{a,b,c,d\},\{a,b\},\{a,c\},\{a,d\}\})$

I_3: $(\{a,b,c,d\},$
$\{\{a,b,c\},\{a,b,d\},\{a,c,d\}\})$

I_4: $(\{a,b,c,d,e\},$
$\{\{a,b,c\},\{a,d,e\},\{a,b\},\{a,c\},\{a,d\},\{a,e\}\})$.

(See figure 3-1.)

Proof:

The hypergraph HC_i is a simple hyperpath with i vertices. Contract all but three edges; the result is a simple hyperpath with three vertices. This hypergraph violates the Helly property and so isn't even a chordal hypergraph, much less a directed path hypergraph.

The hypergraph I_2 has no directed path tree because at most one of the vertices b, c and d can be the parent of the vertex a; the others must be children of a. Since two must be children of a, $\{a,b,c,d\}$ cannot be a path.

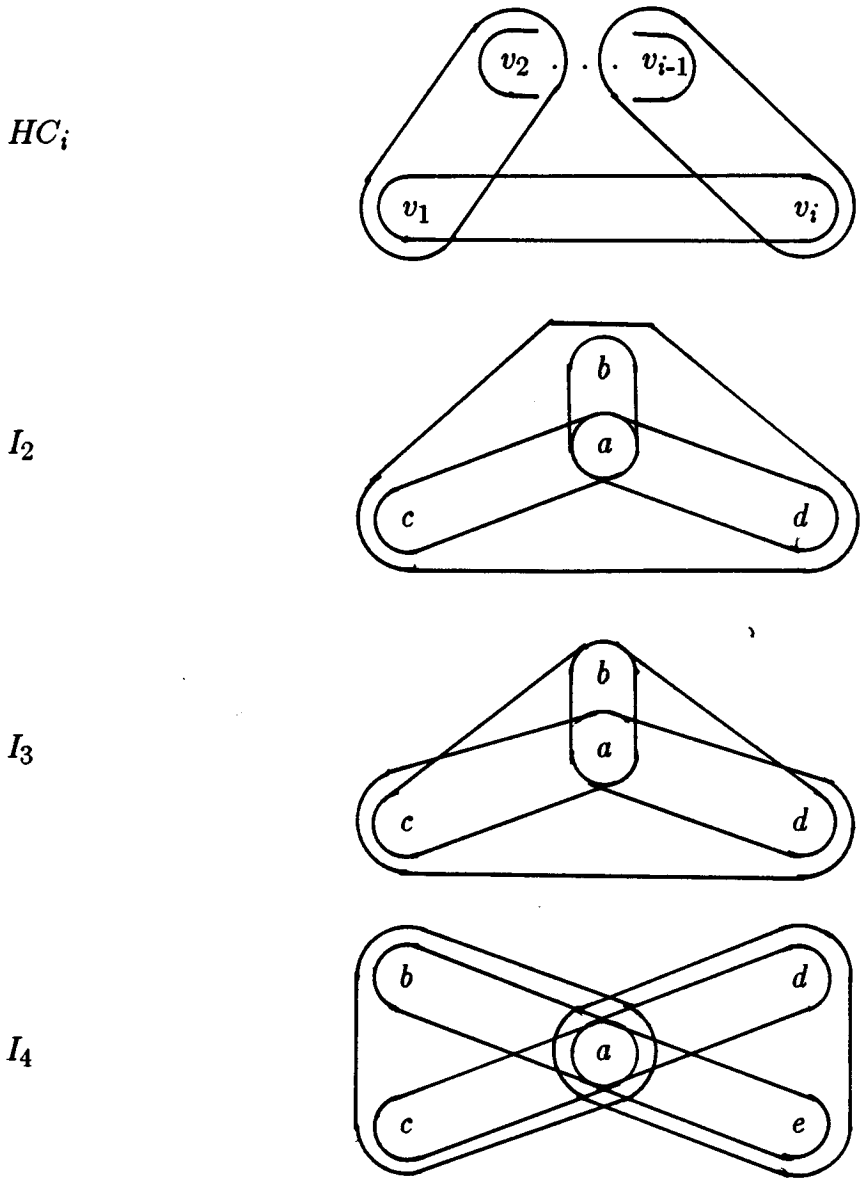

Figure 3-1: Hypergraphs HC_i, I_2, I_3, I_4

Similar reasoning applies to the hypergraphs I_3 and I_4. By lemma 3-1 we can add to I_3 edges $\{a,b\}$, $\{a,c\}$ and $\{a,d\}$, so in any directed path tree at least two of the vertices b, c and d must be children of a, implying that at least one of the edges $\{a,b,c\}$, $\{a,b,d\}$ and $\{a,c,d\}$ is not a path. To see that I_4 is not a directed path hypergraph note that in any supposed directed path tree at most one of the vertices b, c, d and e is not a child of a, so either $\{a,b,c\}$ or $\{a,d,e\}$ is not a path. #

Lemma 3-3:

Let the hypergraph H have directed path tree and let $P=v_0,e_1,\cdots,v_{k-1},e_k,v_k$ be a simple hyperpath in H. If T is a directed path tree for H in which the vertex v_0 is an ancestor of the vertex v_1 then, for $0 \leq i \leq j \leq k$, each v_i is an ancestor of v_j.

Proof:

By induction on k. If $k=0$ or 1, the lemma is trivial.

Suppose $k > 2$. Assume the lemma is true for simple hyperpaths of length less than k. Then, v_i is an ancestor of v_j in T, $0 \leq i \leq j < k$. If v_k is an ancestor of v_{k-1} then either v_k is a descendant of v_{k-2}, in which case e_{k-1} does not induce a directed path in T or v_k is an ancestor of v_{k-2} in which case e_k does not induce a path in T. So, v_k must be a descendant of v_{k-1}. #

3.2. Partial Path Trees

The algorithms presented in this thesis manipulate structures called *partial path trees (PPT's).* If H is a connected hypergraph then

Definition 3-4:

A *partial path tree (PPT)* for H is a directed tree P such that

1. The vertices of P are nonempty disjoint subsets of $V(H)$, $\cup V(P)=V(H)$. (If x is a vertex in $V(H)$, let $P(x)$ denote the vertex in P containing x.)
2. For any vertices u and v in $V(P)$, u a proper ancestor of v, if $e \in Inc_H(u) \cap Inc_H(v)$ then v is a subset of e.
3. If u and v are unrelated vertices in $V(P)$ then $Inc_H(u)$ and $Inc_H(v)$ are disjoint.
4. For any vertices u, v and w in $V(P)$, u the parent of v, v a proper ancestor of w, $Inc_H(u) \cap Inc_H(w)$ is a proper subset of $Inc_H(u) \cap Inc_H(v)$.
5. For any vertex v in $V(P)$, the set

$$CONN(v)=\{e\text{-}Below_P(v) : e \in Inc_H(v),\ e\text{-}Below_P(v)\text{nonempty}\}$$

called the *connector* of v, is linearly ordered by $\subseteq$ (it is a chain), where

$$Below_P(v) = \cup \{v \in V(P) : v \text{ is a descendant of } u \text{ in } P\}.$$

6. The root of P is a directed path in any directed path tree for H.

The definition of PPT's is quite complicated. It has been chosen so that the following lemma holds.

Theorem 3-5:

For any connected DPH H, there is at most one PPT with a given root.

Proof:

The proof is by induction on the number of vertices in the hypergraph H.

If H has only one vertex then H has exactly one PPT.

Assume the lemma is true for all hypergraphs containing fewer than $|V(H)|$ vertices, $|V(H)| < 1$. Let P and P' be PPT's for H with $root(P)=root(P')=x$.

Let s and r be distinct vertices in some child of x in P. By part 2 of definition 3-4, $Inc_H(x) \cap Inc_H(s)=Inc_H(x) \cap Inc_H(r)$, and, by parts 3 and 4, this set is nonempty (since H is connected). Therefore, s and r cannot be in unrelated vertices in P'. But both s and r must be in a child of x in P'; otherwise, if, say, $P'(s)$ is not a child of x then $Inc_H(x) \cap Inc_H(s)$ is not a proper subset of $Inc_H(x) \cap Inc_H(y)$, where

y is the child of x between x and $P'(s)$. So, s and r must be in the same child of x in P'. Similarly, if s and r are in the same child of x in P' they must be in the same child of x in P. We conclude the x has the same children in P and P'.

Since P is connected, if y is a child of x in P and P' then $Below_P(y)=Below_{P'}(y)$. Since the subtrees of P and P' rooted at y are PPT's for the hypergraph $H[Below(y)]$, by induction these trees are identical. Therefore, the trees P and P' are identical. #

We now prove that every directed path hypergraph has a PPT. We say that a PPT P for H *approximates* a directed path tree T if, for any vertices a and b in $V(H)$, if a is an ancestor of b in T then $P(a)$ is an ancestor of $P(b)$ in P.

Lemma 3-6:

For any directed path tree T for H there is a unique partial path tree for H with root $\{root(T)\}$, and this PPT approximates T.

Proof:

There is at most one such PPT (lemma 3-5).

To show that there is at least one such PPT we use induction on $|V(H)|$.

If $|V(H)|=1$, then the directed tree $P=(\{root(T)\}, \emptyset)$ approximates T.

Suppose $|V(H)| > 1$. Assume the lemma is true for hypergraphs with fewer vertices. Choose some leaf a of T. By lemma 3-1, the tree T'=T-$\{a\}$ is a directed path tree for H-$\{a\}$. By the induction hypothesis, there is a PPT P' with root $root(T')$ (which equals $root(T)$) approximating T'. Let b be the parent of a in T. If b=$root(T)$ then we may add a new child $\{a\}$ below $\{b\}$ in P' to obtain a PPT for H. If $b \neq root(T)$ there are two cases. Let s be b's parent in T. If $Inc_H(s) \cap Inc_H(b) = Inc_H(s) \cap Inc_H(a)$ then adding a to $P'(b)$ yields a PPT for H approximating T. Otherwise, adding a child $\{a\}$ below $P'(b)$ yields a PPT for H approximating T. #

Let P be a PPT for H, x a vertex in P. Define

$$m(x) = \begin{cases} x \text{ if } x=root(T) \\ \cap (Inc_H(x) \cap Inc_H(y)) \text{ otherwise,} \\ \qquad\qquad \text{where } y \text{ is the parent of } x. \end{cases}$$

$$M(x) = \{v : c \in \cup Inc_H(x),\ P(v) \text{ an ancestor of } x\}.$$

(See figure 3-2.)

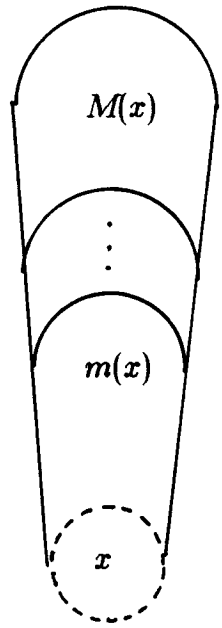

Figure 3-2: $m(x)$ and $M(x)$

Lemma 3-7:

Let P be a PPT for a connected hypergraph H, and let x be a vertex in P.

1. The vertex x is a subset of the sets $m(x)$ and $M(x)$, with equality if and only if x is the root of P.
2. The sets $m(x)$ and $M(x)$ intersect only ancestors of x in P, and
3. The sets $m(x)$ and $M(x)$ induce directed paths in any directed path tree for H.

Proof:

If x is not the root of P, let y be x's parent.

(1) Since $Inc_H(x) \cap Inc_H(y) = Inc_H(r) \cap Inc_H(y)$ for every r in x, x is a subset of $m(x)$. Since H is connected, part 4 of definition 3-4 implies that $Inc_H(x) \cap Inc_H(y)$ is nonempty, so $m(x)=x$ iff x is the root of P. Similarly, if x is not the root then $M(x)$ intersects y (trivially, x is a subset of $M(x)$).

(2) This is also trivial for $M(x)$. For $m(x)$, if x is not the root and z is a proper descendant of x then part 4 of definition 3-4 states that $Inc_H(z) \cap Inc_H(y)$ is a proper subset of $Inc_H(x) \cap Inc_H(y)$, so $m(x)= \cap (Inc_H(x) \cap Inc_H(y))$ and z are disjoint.

(3) Since $m(x)$ is either the root or the intersection of edges in $E(H)$, by lemma 3-1 $m(x)$ induces a path in any directed path tree for H. To prove that $M(x)$ induces a path, observe that there must be some edge e in $Inc_H(x)$ such that $M(x) \subseteq e$ (this follows from definition 3-4, parts 4 and 5). Add edge $m(v) \cap e$ to H for every ancestor v of x for which this set is nonempty. By part (1) of this lemma, each $m(v)$ intersects $m(w)$, w the parent of v, and since the union of these sets is $M(x)$, by lemma 3-1 $M(x)$ induces a path in any directed path tree for H. #

Let the connector of a vertex x, $CONN(x)$, be the set $\{c(x,1), \cdots, c(x,k)\}$, where $c(x,i) \subset c(x,i+1)$. Define

$$d(x,i) = \begin{cases} c(x,1), \text{ if } i=1 \\ c(x,i)\text{-}c(x,i\text{-}1), \text{ if } 1 < i \leq |CONN(x)| \end{cases}$$

and

$$e(x,i) = \begin{cases} c(x,|CONN(x)|) \text{ if } i=1. \\ c(x,|CONN(x)|)\text{-}c(x,i\text{-}1) \text{ if } 1 < i \leq |CONN(x)| \end{cases}$$

(See figure 3-3.)

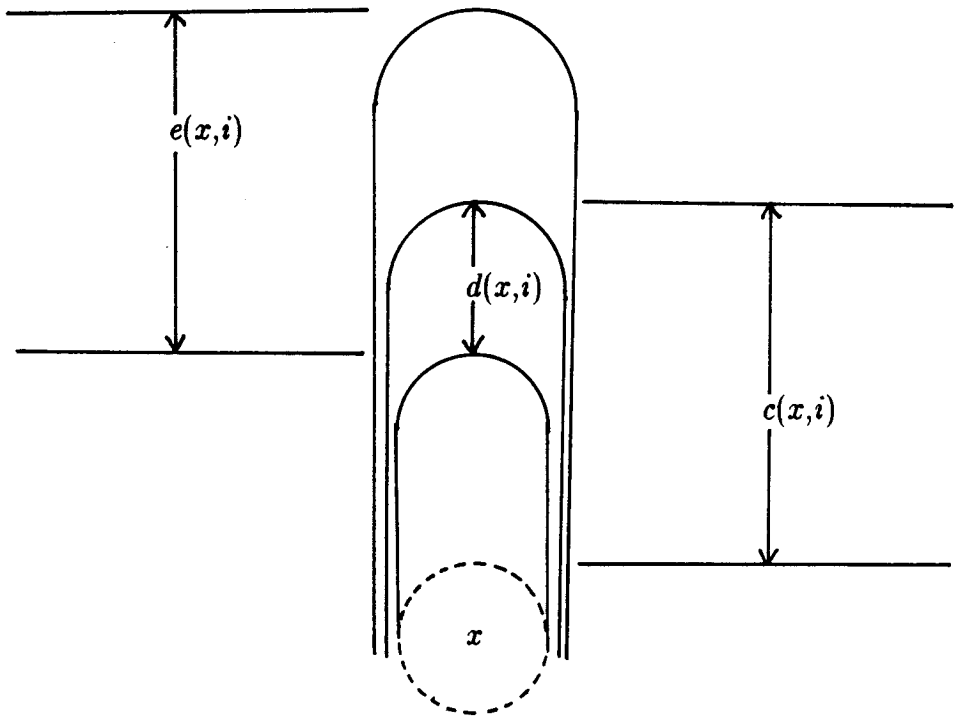

Figure 3-3: Sets $c(x,i)$, $d(x,i)$, $e(x,i)$

Note that $m(x)=x \cup c(x,1)$ and $M(x)=x \cup e(x,1)$.

Lemma 3-8:

Let P be a PPT for H, x a vertex in P. Each set $c(x,i)$, $d(x,i)$ and $e(x,i)$ induces a directed path in any path tree for H. Moreover, if $|CONN(x)| > 1$ then the vertices in node x induce a directed path in any path tree for H.

Proof:

Let T be a path tree for H. If x is the root, the lemma is vacuously true. If x is not the root, let y be the parent of x in P. We know that each $c(x,i)$ is a set of the form $e \cap M(y)$, e an edge in $E(H)$ intersecting both x and y, and therefore induces a directed path in T.

Similarly, $c(x,i)$ and $c(x,i\text{-}1)$ are subsets of $M(x)$ and $d(x,i)=c(x,i)\text{-}e$ for some edge in $E(H)$ intersecting both x and y. Therefore, $c(x,i)$ and $c \cap M(x)$ are intersecting subsets of $M(x)$ but neither is a subset of the other, so (lemma 3-1, part 6) $d(x,i)$ induces a path in T. A nearly identical argument also works for the $e(x,i)$'s and also shows that, if $|CONN(x)| > 1$ then x induces a path in T. #

Lemma 3-9:

If $|CONN(x)| > 1$ then in any directed path tree T for H the vertices in $d(x,i)$ are ancestors of those in $d(x,i+1)$ $(1 \leq i < |CONN(x)|)$ or the vertices in $d(x,i)$ are descendants of of those in $d(x,i+1)$ $(1 \leq i < |CONN(x)|)$.

Proof:

We know that the vertices in

$$x \cup d(x,1) \cup \cdots \cup d(x,|CONN(x)|) = M(x)$$

induce a path in T (lemma 3-7). The sets $x \cup d(x,1)$ $(=m(x))$ and $d(x,i) \cup d(x,i+1)$ $(=c(x,i+1) \cap e(x,i)$, $1 \leq i < |CONN(x)|)$ induce paths in T (lemmas 3-1, 3-7 and 3-8). Lemma 3-3 then proves the lemma. #

Lemma 3-10:

Let P be a PPT for H. If H has a directed path tree then P is a PPT for H^*, where H^* is the hypergraph obtained by repeatedly applying operations (2) and (5) to H (called the *closure* of H).

Proof:

We verify that, after applying operation (2) or (5) to H, P is still a PPT for the new hypergraph H'. There are six possibilities, corresponding to the six parts of definition 3-4.

(1) is trivial.

(2) remains true when adding the intersections of edges: let $u,v \in V(P)$, u a proper ancestor of v. Let c and d be edges in $E(H)$ intersecting both u and v. Since v is a subset of c and d, it is a subset of their intersection. (2) also remains true when joining edges. Let c and d intersect u and v, respectively, and let the intersection of c and d be nonempty. Either d intersects v's parent, in which case v is a subset of

d, or c intersect v, in which case v is a subset of c. In either case v is a subset of the union of c and d.

(3) remains true because c and d cannot create a new edge that is not a subset of previously existing edge.

(4) can only be invalidated if some edge is created that intersects u and w but not v, u the parent of v, v a proper ancestor of w. Intersecting edges cannot do this, because if edges c and d intersect u and w then v is a subset of the intersection of c and d. Joining edges cannot do it, because if edge c intersects u and v intersects w and c intersects d then either c intersects v (and v is a subset of c) or d intersects u (and v is a subset of d).

(5) If (5) becomes false, then we can simplify H' to HC_3, which has no directed path tree, so H' and therefore H have no directed path trees (lemmas 3-1, 3-2, 3-7). Since we assumed that H has a directed path tree this can't happen.

(6) is trivial.

Since a single application of operations (2) or (5) preserves PPT's, so do sequences of these operations, so P is a PPT for H^*. #

Lemma 3-11:

Let P be a PPT for H^*, and let u and v be vertices in P, u an ancestor of v. For any $r \in u$ and $s \in v$ there is a simple (r,s) hyperpath in H^* containing edges intersecting only ancestors of v.

Proof:

Let $u=v_1, \cdots, v_k=v$ be the (u,v) directed path in P. We can build a (not necessarily simple) (r,s) hyperpath using the edges $m(v_i)$, $1 \leq i \leq k$ and by selecting one element from each v_i (select r and s from u and v). By lemma 3-7 these edges intersect only ancestors of v. Since every (r,s) hyperpath has as a subhypergraph a simple (r,s) hyperpath, there is a simple (r,s) hyperpath in H^* with edges intersecting only ancestors of v. #

4. Algorithms on Partial Path Trees

This chapter describes three algorithms on partial path trees. The first, called *Find-PPT-with-Root*, determines if a hypergraph H has a PPT with a given root r, where r is a subset of the set of vertices $V(H)$ that induces a path in any directed path tree for H. The second, *Find-PPT*, will produce a PPT for H if H has a directed path tree. The final algorithm, *Find-Path-Tree*, takes as input a PPT for hypergraph H and produces a directed path tree for H, if one exists. All of these algorithms run in linear (that is, $O(|V|+SIZE(E(H)))$ time.

4.1. Finding a Partial Path Tree with a Given Root

We now give an algorithm that, given a root $r \subseteq V(H)$, will construct a PPT P for H with root r (if one exists). The algorithm runs in linear time. By lemma 3-6, if H has a directed path tree with root r in $V(H)$ then H has a PPT with root $\{r\}$. So, if there is no r such that H has a PPT with root $\{r\}$ then H has no directed path tree. Combined with the algorithm from section 4.3 one gets a quadratic time algorithm for finding directed path trees for directed path hypergraphs. This is essentially Truszczynski's algorithm [44].

The tree is constructed in a depth-first manner. When a vertex v of the PPT is visited all edges in $E(H)$ intersecting v that have not yet been added to the tree are added. This may alter the subtree rooted at v, but not v itself. At each stage the tree is a PPT for the edge induced subhypergraph $H[E']$, where E' is the set of edges so far added.

Figures 4-1 and 4-2 present the details of the algorithm.

Lemma 4-1 proves the partial correctness of *Find-PPT-with-Root*.

We say that a vertex x has been *visited* if the procedure Visit has been called with x as its argument.

Let H be a hypergraph and let r, a vertex in $V(H)$, be the root of some directed path tree for H.

Lemma 4-1:

After each iteration of the loop at line 5 in procedure Visit (figure 4-1), the tree P is a PPT for $H[E']$, and if the algorithm halts without aborting $E'=E(H)$, assuming H is connected.

Proof:

By induction on $|E'|$. Initially, $E'=\{r\}$ and P is the tree with the single vertex r, so P is a PPT for $H[E']$.

Assume the lemma is true for $1 \leq |E'| < n$ $(n > 1)$. Let x be the vertex being visited, e the edge being added. Since vertices are visited in preorder no descendant of x has yet been visited, so for every descendant y of x, $Inc_H(y) \cap E' \subseteq Inc_H(x) \cap E'$.

Let d be $c \cap V'$. If the set d induces a disconnected subgraph of

Comment Construct a PPT for H=(V,E) with root $r \in E$;

proc *Find-PPT-With-Root*(V,E,r)

begin

1. $E' := \{r\}$;
2. $V' := r$;
3. $P := (\{r\}, \emptyset)$;
4. *Visit*(r)

end *Find-PPT-With-Root*;

Comment Visit vertex x, adding all edges in $Inc_H(x)$-E';

proc *Visit*(x)

begin

5. **for** each c in $Inc_H(x)$-E' in decreasing order of $|e \cap x|$ **do**
6. *Add-Edge*(c,x);
7. $E', V' := E' \cup \{c\}, V' \cup c$
8. **for** each child s of x **do** *Visit*(s)

end *Visit*;

Figure 4-1: Procedures *Find-PPT-With-Root* and *Visit*

Comment Add an edge c below vertex x in PPT P;

proc *Add-Edge*(c,x)

begin

9. $d,e := c \cap V', c - V'$;

10. **if** d is not a subset of x **then**

11. $L := \{s : s$ is a vertex in P, s intersects c, s has no proper descendants in $L\}$;

12. **if** L contains a nondescendant of x or $|L| > 1$ **then abort**;

13. **if** $|L| = 1$ **then**

14. $u_0, \ldots, u_m :=$ the (x,u) directed path in P, where $L = \{u\}$;

15. **if** u_i is not a subset of c for some i, $1 \leq i < m$ **then abort**;

16. $b :=$ the most recently added edge intersecting u_1;

17. **if** $c \cap x$ is not a subset of $b \cap x$ **then abort**;

18. **if** u is not a subset of d **then**

19. $y := u - d$;

20. Add vertex y to P;

21. Make every child of u in P a child of y;

22. Replace u with $u \cap d$ in P;

23. Make y a child of $u \cap d$ in P

end *Add-Edge*

Figure 4-2: Procedure *Add-Edge*

$H[E']$ then the hypergraph $H[E' \cup \{c\}]$ has a simple hypercycle of length at least three, an impossibility since H has a directed path tree. Therefore, H^* contains the edge

$$\cup \{e \cap c : e \in Inc_H(c) \cap E'\}$$

(lemma 3-1), which is d.

So, we add d to the tree. There are two cases. If d intersects only x the tree is unchanged. Otherwise, d intersects a sequence of vertices $x = u_0, u_1, \ldots, u_m$, u_{i-1} the parent of u_i in P and $u_1, \ldots, u_{m-1}$ subsets of c. If u_m is also a subset of c then P is a PPT for $H[E' \cup \{d\}]$. Otherwise, the algorithm splits u_m into $u_m \cap d$ and u_m-d, making the first the parent of the second. All children of u_m become children of u_m-d. It is not difficult to verify that the result is a PPT for $H[E' \cup \{d\}]$ (part 4 of definition 3-4 is the critical part).

Once we have a PPT for $H[E' \cup \{d\}]$ we can get a PPT for $H[E' \cup \{c,d\}]$ by making c-d a child of the deepest vertex intersecting d. By theorem 3-5 and lemma 3-10 the result is also a PPT for $H[E' \cup c]$.

Since H is connected every edge is eventually added, so E' eventually becomes E. #

Lemma 4-2 provides the justification for the crucial step in the linear time

algorithm. It gives conditions under which vertices can be discarded from a hypergraph while building the hypergraph's PPT. The lemma is true because all obstacles that can arise while building a PPT force the root of an directed path tree for H to be at or below the obstacle (see the lemma for a precise formulation).

Lemma 4-2:

Let H be a hypergraph, r an edge in $E(H)$, P a PPT for H with root r and P' a PPT for $H[E']$ with root r constructed by *Find-PPT-with-Root*. Let U be the set of vertices in $V(P')$ visited so far. Then, $U \subseteq V(P)$ and $P'[U]=P[U]$.

Proof:

By induction on the size of U. Initially, $U=\{r\}$, and since r is the root of P, $P[\{r\}]=P'[\{r\}]$.

Assume that $U \subseteq V(P)$ and $U \subseteq V(P')$, $1 \leq |U| < k$. Let y be the k-th vertex in P' to be visited. Let x be y's parent in P'. Since the vertices are visited in preorder, x is in $V(P)$, and all edges in $Inc_H(x)$ are in E'. We can therefore show, by arguments similar to those used in the proof of theorem 3-5, that the children of x are the same in P and P'. Therefore, y is in $V(P)$. #

Finally, we must show that when the algorithm aborts, H has no partial path tree with root r or has no directed path tree whatsoever.

Lemma 4-3:

If *Find-PPT-with-Root* aborts then either H has no PPT or H has no directed path tree.

Proof:

There are four places where the algorithm can abort, all in procedure *Add-Edge*. Let P' be the PPT constructed so far, x the vertex being visited, c the edge being added.

Line 13: The edge c intersects a vertex u in $V(P')$ that is not a descendant of x. The vertex u must be unrelated to x; otherwise, the edge c would have already been added to E'. Let v be the least common ancestor of x and u. There is a simple hyperpath from some vertex in $x \cap c$ to some vertex in $u \cap c$ passing through v. Adding c to this path yields a simple hypercycle of length at least three so (lemmas 3-1, 3-2) H has no directed path tree.

Line 13: The edge c intersects two unrelated descendants of x, say u and v. By lemma 4-2, either H has no directed path tree or the vertices in u and v must be in proper descendants of x in any PPT for H with root r. In the latter case, let s and t be vertices in $u \cap c$, $v \cap c$, respectively. Since the intersections of $Inc_H(s)$ and $Inc_H(t)$ with $Inc_H(x)$ are incomparable, s and t must be in unrelated descendants of x (definition 3-4, part 4), an impossibility (definition 3-4, part 3). So, either H has no directed path tree or H has no PPT with root r.

Line 16: The descendants of x that intersect c lie along a directed path, u the deepest such descendant, and their is a vertex v between x and u that is not a subset of c. Let s and t be vertices in v-c and $u \cap c$, respectively. As in the previous case, lemma 4-2 requires that s and t be in unrelated descendants of x in any PPT for H with root r, an impossibility.

Line 18: Some previously added edge b intersects a child u_1 of x, let $c \cap x$ is not a subset of $b \cap x$. We know that b does not intersect x's parent (if x has a parent); otherwise, $c \cap x$ would be a subset of $b \cap x$. But since edges are added by Visit in decreasing order of the size of their intersections with x, we know that $|b \cap x| \geq |c \cap x|$, so neither $b \cap x$ nor $c \cap c$ is a subset of the other. By deleting from H all but one record from $(b \cap x)$-c, $(c \cap x)$-b and $u_1 \cap c$, we can simplify H to HC_3, so (lemmas 3-1, 3-2) H has no directed path tree. #

Theorem 4-4:

Find-PPT-with-Root is correct.

Proof:

Immediate from lemmas 4-1 and 4-3. #

It should be mentioned that two of the ways the algorithm can abort (the first and last) cannot happen if H is a chordal hypergraph.

We now outline a linear time implementation of this algorithm. Proper choices of data structures are crucial.

The hypergraph $H=(V,E)$ is represented as follows. Each pair (v,e), v a vertex in V, e a vertex in $E(H)$, v an element of e, is represented by a record in memory. These records are strung together into doubly linked lists, one for each vertex v (containing all records of the form $(v, \cdots)$) and one for each edge e (containing all records of the form $(\cdots, e)$). These records can be deleted and inserted in constant time. The vertices of the PPT P are represented by linked lists of vertices from $V(H)$ (each vertex is in at most one tree vertex).

The procedure *Visit* is implemented as follows. Let x be the vertex being visited. For each vertex $r \in x$, traverse the list list of pairs (r,e). For each edge e found, increment by one a counter (initially zero) associated with that edge. After all lists are traversed, which takes time proportional to $|x|$ plus the sum of the sizes of the $Inc_H(r)$'s $(r \in x)$, the counters contain the sizes of the intersections of the edges in $Inc_H(x)$-E' with x. These numbers can be sorted by a simple bucket sort in $O(|x|+|Inc_H(x)\text{-}E'|)$ time. In decreasing order of $|e \cap x|$ we add these edges to P (see below) and delete them from C. This takes time $|e|$ per edge e.

Add-Edge is also straightforward. First, partition the edge c into $c \cap V'$ and c-V'. This takes $O(|c|)$ steps. Finding the vertices in P that intersect c can also be done in $O(|c|)$ steps. Call this set of vertices D. If the parent of any vertex in D-$\{x\}$ is not in D then stop and abort. If any vertex in D-$\{x\}$ that has a child in

D is not a subset of c then stop and abort. Finally, if there is more than one vertex in D that has no child in D then stop and abort. This all takes $O(|c|)$ steps.

For every unvisited vertex in $V(P)$ we keep track of the last edge added to the tree that intersected v, and store the intersection of b and v. This took $O(|b|)$ steps when b was added. When c is added, comparing $c \cap x$ and $b \cap x$ takes $O(|c|)$ steps. Splitting the deepest vertex in D and adding c-V takes $O(|c|)$ steps. Since every part of *Add-Edge* takes at most $O(|c|)$ steps, *Add-Edge* takes $O(|c|)$ steps.

Adding up the running time gives:

Proposition 4-5:

Assuming H is connected, algorithm *Find-PPT-with-Root* returns or aborts within $O(SIZE(E(H)))$ steps.

We can find a PPT for any connected DPH H by running *Find-PPT-with-Root* for $r=\{v\}$ for each vertex v in $V(H)$. By lemma 3-6, there must be some vertex v in $V(H)$ for which the algorithm will succeed. Worst case running time will be $O(|V|+SIZE(E))$.

4.2. Finding a Partial Path Tree -- General Algorithm

In the previous section we saw how to determine in linear time if a hypergraph has a PPT with a given root. In general, however, we do not know ahead of time what the root will be. This section describes a linear time algorithm for finding such roots.

The algorithm begins by building a PPT for the hypergraph H at some arbitrary root $\{r\}$, r a vertex in $V(H)$. When obstacles are encountered the algorithm discards portions of the tree and chooses a new root. This process continues until a PPT P' for a subhypergraph H' of H has been found. This subhypergraph has the property that the existence of a directed path tree for H guarantees the existence of a PPT for H with root $root(P')$. If the algorithm fails then H has no directed path tree.

We begin by proving several lemmas that are crucial to understanding the linear time algorithm.

First, lemma 4-6 shows how to simplify a PPT for a hypergraph H when a vertex is deleted from H.

Let H be a hypergraph, r a vertex in $V(H)$, $H'=H-\{v\}$. As usual, assume H and H' are connected, and that $V(H')$ is nonempty. Let P be a PPT for H. Build a PPT P' for H' as follows:

1. Delete r from $P(r)$. If $P(r)=\{r\}$ and $P(r)$ is the root then delete it and, since P' is connected, $P(r)$ has only one child; it becomes the root of P'. Otherwise, if $P(r)=\{r\}$ and $P(r)$ is not the root then $P(r)$'s parent inherits $P(r)$'s children.
2. While there are vertices x,y and z in the tree, y a child of x, z a child of y, $Inc_{H'}(x) \cap Inc_{H'}(y) = Inc_{H'}(x) \cap Inc_{H'}(z)$, replace y and z by their union. All children of y (except z) and all children of z become children of the next vertex.

Lemma 4-6:

P' is a PPT for H'.

Proof:

Observe that the tree obtained after step (1) satisfies all parts of definition 3-4 except part 4. As vertices are merged in step (2) this remains true. When this loop concludes (as it must, since their are a finite number of vertices) part 4 of definition 3-4 is also satisfied, and the tree is a PPT for H'. #

Corollary 4-7:

If $v_1, \ldots, v_k \in V(H)$ are not in $root(P)$ then $H\text{-}\{v_1, \ldots, v_k\}$ has a PPT with root $root(P)$.

The next lemma (lemma 4-8 gives conditions under which the existence of a PPT for a subhypergraph $H[V']$ of H implies the existence of a PPT for P (with the same root). This lemma is central to the linear time algorithm; it will allow use to discard portions of a hypergraph, find a PPT for the remaining subhypergraph, then extend this PPT to a PPT for the entire hypergraph.

Let $H=(V,E)$ be a hypergraph, $V' \subseteq V$, $H'=H[V']$. Assume H and H' are connected.

Lemma 4-8:

If P' is a PPT for H' with root v, v an edge in E, and if there is a directed path tree T for H in which all vertices in $\cup Inc_H(V\text{-}V')$ are descendants of some vertex r in v then H has a PPT P with root v.

Proof:

We show that there is a PPT P with root v by running *Find-PPT-with-Root* on H, starting at vertex v. We prove that it cannot abort so (theorem 4-4) it must return a PPT for H with root v.

To show that the algorithm cannot abort, we consider each of the four cases that could cause it to fail. Two of those cases ($c \cap x$ is not a subset of $b \cap x$, and c intersects a nondescendant of x) imply that H has no directed path tree, a contradiction. We consider the other two cases. Let x be the vertex being visited and c the edge being added.

At line 13, suppose there are two unrelated descendants of x, say y and z, that intersect c. Let b and d be the most recently added edges intersecting y and z (other than c). Since H has a directed path tree, $c \cap x$ must be a subset of $b \cap x$ and $d \cap x$. Let $r_1 \in y \cap c$, $r_2 \in z \cap c$ and $r_3 \in x \cap c$. By lemma 3-1, in any path tree for H exactly one of r_1 or r_2 is an ancestor of r_3. Without loss of generality, assume it is r_1 in the directed path tree T. By lemmas 3-11 and 3-3 there is in H^* a simple (r_3,s)-hyperpath (s any vertex in v) that does not contain edges

intersecting y or z and therefore r_1 is an ancestor in T of all vertices in v. So, r_1 is not in any edge intersecting V-V', so c is a subset of V'.

If x is the root then H' has no PPT with root v, since r_1 and r_2 must be in unrelated descendants of v, and $r_1 \in b$-d, $r_2 \in d$-b and $b,d \in Inc_H(v)$, yet both $r_1,r_2 \in c$ (violating definition 3-4, part 3).

So, $x \neq v$. Let $v=x_1, \ldots, x_n=x$ be the (v,x)-directed path in P'. We show that there is a subset of V' (of which v is a subset) that induces a connected subhypergraph of P' that has no PPT with root v, a contradiction (corollary 4-7).

The set c is disjoint from $\cup Inc_H(V$-$V')$, because $r_1 \in z$, c is disjoint from v, and r_1 is an ancestor in T of all vertices in v (this follows from lemma 3-1: contract c; the vertex v_c must be a proper ancestor of all vertices in v). So, no edge intersecting x and c intersects V-V', and therefore $M(x_n)$ is a subset of V'. If x_i is the deepest vertex along to (v,x)-directed path that intersects an edge intersecting v then for all j, $i < j \leq n$, the vertices in $M(x_j)$ are ancestors in T of those in v (lemmas 3-1, 3-3 and 3-11), so $M(x_j)$ does not intersect any edge intersecting V-V'.

If there exists an edge intersecting both x_n and v then let the set X be the union of $x_1, \ldots, x_n$ and $\{r_1,r_2\}$. We know that each x_i is a subset of $M(x_n)$. Is there a PPT for $H'[X]$ with root v? No: if there were then

(lemma 4-6) the vertices $x_1, \ldots, x_n$ would exist in it as well, and r_1 and r_2 would necessarily lie in unrelated descendants of x_n. This again contradicts definition 3-4, part 3.

So, assume that there is no edge intersecting both x_n and v. Let x_i be as above; let X be $v \cup M(x_{i+1}) \cup \ldots \cup M(x_n) \cup \{r_1, r_2\}$. By lemma 4-6, a PPT for $H'[X]$ with root v would contain a directed path from v, the union of the vertices in the path equal to $(x_1 \cup \ldots \cup x_i) \cap M(x_{i+1})$. The deepest vertex in this directed path would include all vertices in $M(x_{i+1}) \cap x_i$, so the vertices $x_{i+1}, \ldots, x_n$ would be unchanged. Again, r_1 and r_2 must be in unrelated descendants of x_n, a contradiction.

This exhausts line 13, so *Add-Edge* cannot abort there.

Line 16: Here, the edge c intersects some proper descendants of x, say y and z, y a proper ancestor of z, y not a subset of c. Let $r_1 \in z \cap c$, $r_2 \in y\text{-}c$, $r_3 \in x \cap c$. Let b be the most recently added edge (before c) intersecting z and d the most recently added edge intersecting y but not z. As before, $c \cap x$ is a subset of $b \cap x$ and $d \cap x$. We can contract the edge $M(x)$ and delete all other vertices except r_1 and r_2 and conclude (lemmas 3-1, 3-7 and 3-3) that either r_1 or r_2 is an ancestor in T of all vertices in v, so b and d are subsets of V'. From this point on the proof is the same as the previous case.

Since *Find-PPT-with-Root* does not abort on root v, H has a PPT with root v. #

Figures 4-3, 4-4, 4-5, 4-6, 4-7 and 4-8 give the details of the linear time algorithm. Let $Above(x,c) = M(x)\text{-}(x \cap c)$.

The next theorem shows that if this algorithm aborts then H has no directed path tree.

Theorem 4-9:

Let P_0 be a PPT for $H' = H[E']$, $E' \subseteq E$. Let x be a vertex in P_0 such that for every descendant v of x, the edges in E' intersecting v also intersect x. Let c be an edge in $E\text{-}E'$ intersecting x that is being added to the tree by *Add-Edge* (figure 4-4) at vertex x. If *Add-Edge* aborts then H has no directed path tree.

Proof:

There are five places where the algorithm can abort.

(Line 10) If c intersects some vertex y in P_0 unrelated to x then H has a simple hypercycle, so H has no directed path tree.

(Line 12) If c intersects at least three unrelated descendants of x, say y_1, y_2 and y_3, then there are edges e_1, e_2 and e_3 in E' such that each e_i intersects x and y_i but not the other y_j $(i \neq j)$. Let $r_i \in y_i \cap c$,

Comment Build a PPT for H=(V,E) or find that H is not DPH;

proc *Find-PPT*(V,E)

begin

1. $P := (\{e\}, \emptyset)$ for some $e \in E$;
2. V',*Root*,E',$D := e,e,\{e\},\emptyset$;
3. *Visit*(*Root*);
4. **return** (*Find-PPT-with-Root* (R,C,*Root*))

end *Find-PPT*;

Comment Visit the vertex x in P', adding all edges in E-E' intersecting x;

proc *Visit*(x)

begin

5. **for** each c in E-E' intersecting x in decreasing order of $|c \cap x|$ **do**
6. $\quad x := Add\text{-}Edge(c,x)$;
7. **while** x is a vertex in P and x has a child s that hasn't been visted **do**
8. $\quad Visit(s)$

end *Visit*;

Figure 4-3: Procedures *Find-PPT* and *Visit*

Comment Add the edge c to E' while visiting vertex x;

proc *Add-Edge*(c,x)

begin

9. $A := \{v : v$ a vertex intersecting c, $v \neq x\}$;

10. **if** A contains any vertex unrelated to x **then abort**;

11. $L := \{v \in A :$ No proper descendant of v is in $A\}$;

12. **if** $|L| > 2$ **then abort**;

13. **if** c is not a subset of V' **then** $L := Split(c,L)$;

14. **if** $|L| = 1$ **then** $x := Case1(c,x,L)$

15. **else if** $|L| = 2$ **then** $x := Case2(c,x,L)$;

16. $V' := V' \cup c$;

17. $E' := E' \cup \{c\}$;

18. **if** $c\text{-} \cup A$ is not empty **then**

19. Make $c\text{-} \cup A$ a child of the deepest vertex in P intersecting c;

20. **return**(x)

end *Add-Edge*;

Figure 4-4: Procedure *Add-Edge*

Comment Split the vertices in PPT P' which intersect c, have no descendants intersecting c, and are not subsets of c into parts intersecting c and disjoint from c;

proc *Split*(c,L)

begin

21. $N := \emptyset$;

22. **for** each v in L **do**

23. **if** v is a subset of c **then** $N := N \cup \{v\}$

24. **else**

25. Replace v with $v \cap c$ in P;

26. $V' := V' \cup \{m(v) \cap c\}$;

27. Make v-c a child of $v \cap c$;

28. $N := N \cup \{v \cap c\}$

29. **return**(N)

end *Split*;

Figure 4-5: Procedure *Split*

proc *Case*1(c,x,L)

begin

30. $u :=$ the single vertex in "L";

31. $x,u_1, \ldots ,u_m :=$ the path from x to u in P;

32. $e :=$ the most recently added edge in E' intersecting u_1;

33. **if** $c \cap x$ is not a subset of $e \cap x$ **then abort**;

34. **if** each u_i, $1 \leq i < m$, is a subset of c **then return**(x);

35. $s := min\{i : u_i$ is not a subset of $c\}$;

36. **return**(*New-Root* ($M(u_s)$));

end *Case*1;

Figure 4-6: Procedure *Case*1

$1 \leq i \leq 3$. Obtain a hypergraph I from H' by contracting $m(x)$ and deleting all other vertices except the r_i's. The resulting hypergraph is isomorphic to I_2 so (lemmas 3-1, 3-2) H has no directed path tree. See figure 4-9.

(Lines 33) All vertices intersecting c are related, u the deepest vertex intersecting c, e the edge in E' intersecting x and u with the smallest intersection with x, and $c \cap x$ is not a subset of $e \cap x$. In this case we can delete from H all but one vertex in $(c \cap x)$-e, $(e \cap x)$-c (which must be nonempty, since $|e \cap x| \geq |c \cap x|$, and one from $u \cap c$. The resulting hypergraph is isomorphic to HC_3, so H has no directed path tree.

proc *Case2*(c,x,L)

begin

37. u,v := the two vertices in L;

38. x,u_1, . . . ,u_m and x,v_1, . . . ,v_n := the paths from x to u and x to v in P;

39. $k := min\{i : u_i \neq v_i\}$;

40. e,f := the most recently added edges in E' intersecting u_1 and v_1;

41. **if** $c \cap x$ is not a subset of e or of f **then abort**;

42. $b := c$;

43. **if** u_i is not a subset of c for some i, $1 \leq i < m$ or

    there is an edge in E' intersecting u_k and *Above*(x,c) **then** $b := M(u_m)$;

44. **if** v_j is not a subset of c for some j, $1 \leq j < n$ or

    there is an edge in E' intersecting v_k and *Above*(x,c) **then**

45. **if** $b \neq c$ **then abort else** $b := M(v_n)$;

46. **return**(*New-Root* (b))

end *Case2*;

Figure 4-7: Procdure *Case2*

(Line 41) is, with small changes, identical to the previous case.

(Line 45) If the algorithm aborts here then it must have already set b to $M(u_m)$ at the previous statement. Let u and v be the deepest descendants of x intersecting c, and let x,u_1, . . . ,u_m=u and x,v_1, . . . ,v_n=v be the directed paths from x to u and v in P_0, and let k

Comment Create a new root, discarding all vertices above it;

proc *New-Root* (b)

begin

47. $Deep := \{v : v$ a vertex in P, v has an ancestor intersecting b,
    v has no proper descendants that intersect b $\}$;
48. $D := D \cup (\{r \in V' : r$ is not in a vertex in $Deep\}$-$b)$;
49. $D := D \cup \{r \in V' :$ there is a path from r to some vertex in D
    that does not pass through V'-$D\}$;
50. Remove all vertices in D from all edges;
51. $V' := V'$-D;
52. Replace the vertices intersecting b in P with the single vertex b;
53. $Root,E,E' := b,E \cup \{b\},E' \cup \{b\}$;
54. **return**($Root$)

end *New-Root*

Figure 4-8: Procedure *New-Root*

be the smallest number (> 0) such that $u_k \neq v_k$. There are several cases to consider.

1. u_i ($=v_i$) is not a subset of c for some i, $1 \leq i < k$. Let $q \in c \cap x$, $r \in u_i$-c, $s \in u \cap c$ and $t \in v \cap c$. The subhypergraph $H[\{q,r,s,t\}]$ simplifies to I_3, so (lemmas 3-1, 3-2) H has no directed path tree. See figure 4-10.

2. u_i is not a subset of c, v_j is not a subset of c, for some i, j, $k \leq i < m$, $k \leq j < n$. Let $q \in u_i$-c, $r \in v_j$-c, $s \in u \cap c$ and $t \in v \cap c$. The hypergraph obtained from H by adding edges $M(u) \cap c$ and $M(v) \cap c$, contracting $m(x)$ and deleting all other vertices except q,r,s and t simplifies to I_4 so (lemmas 3-1, 3-7 and 3-2) H has no directed path tree. See figure 4-11.
3. There is an edge d_1 in E' intersecting u_k and *Above*(x,c), and an edge d_2 in E' intersecting v_k and *Above*(x,c). We can assume that one of u_k or v_k is a subset of c; otherwise, the previous case applies. Without loss of generality assume u_k is a subset of c. We know that $d_1 \cap d_2 \cap$ *Above*(x,c) is not empty (from definition 3-4, part 5). Let $q \in x \cap c$, $r \in v \cap c$, $s \in u_k \cap c$ and $t \in d_1 \cap d_2 \cap \in$ *Above*(x,c). If v_k intersects c then let $p \in v_k \cap c$, and the hypergraph $H[\{p,q,s,t\}]$ simplifies to I_3. If v_k and c are disjoint then let $p \in v_k$. The hypergraph $H[\{p,q,r,s,t\}]$ simplifies to I_4. In either case H has no directed path tree (lemmas 3-1, 3-2). See figure 4-12.
4. u_i is not a subset of c for some i, $k \leq i < m$, no edge in E' intersects both u_i and *Above*(x,c), and there is some d in $v_k \cap$ *Above*(x,c) (the symmetric case is identical; interchange u and v and m and n in the following.) Assume v_k is a subset of c; otherwise, case 2 applies. Let $p \in v_k$, $q \in u_i$-c, $r \in u \cap c$, $s \in x \cap c$ and $t \in d \cap$ *Above*(x,c). The hypergraph $H[\{p,q,r,s,t\}]$ again

simplifies to I_4, so H has no directed path tree (lemmas 3-1, 3-2). See figure 4-13.

This covers all cases, so if *Add-Edge*, *Case*1 or *Case*2 abort then H has no directed path tree. #

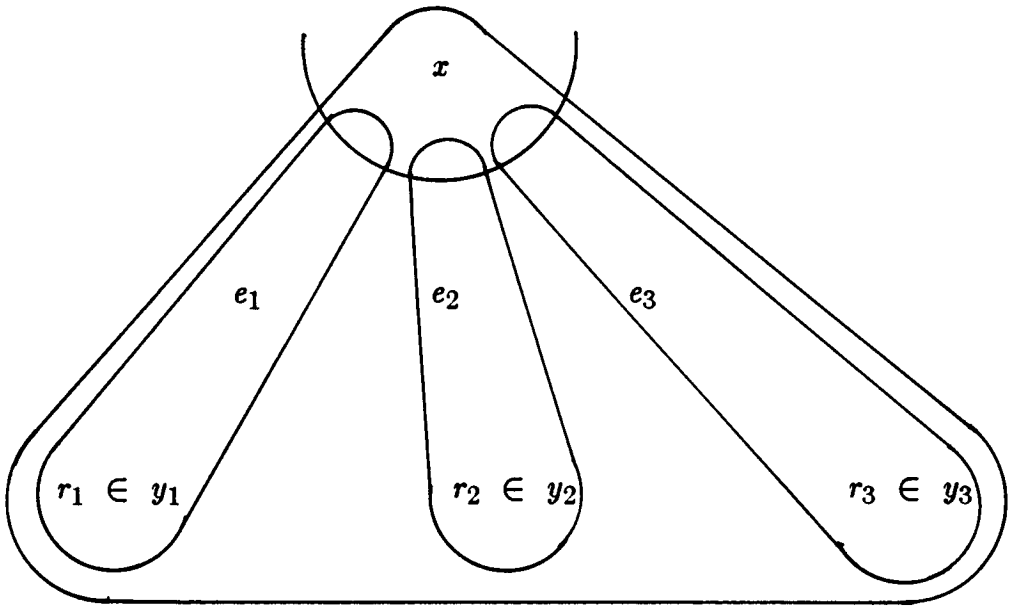

Figure 4-9: Proof of theorem 4-9, line 12

There is another place the algorithm can abort that hasn't been covered. After producing a PPT P for the subhypergraph $H[E']$ the algorithm runs *Find-PPT-with-Root* on H starting on *root*(P). Using lemma 4-8, we can show that if *Find-PPT-with-Root* aborts then H has no directed path tree. To do this, we will show that if *Find-PPT* discards vertices D then in any directed path tree for H the vertices in D must be descendants of some vertex in *root*(P). The next lemma (lemma 4-10 shows that this condition and two others are invariant.

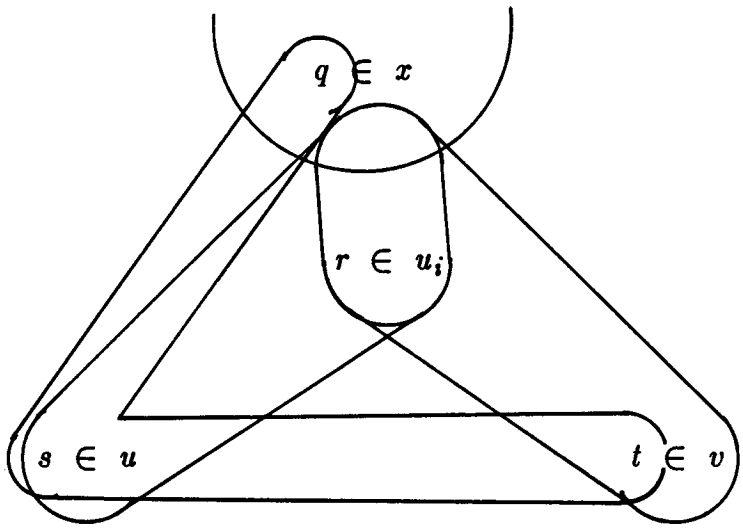

Figure 4-10: Proof of theorem 4-9, line 45, case 1

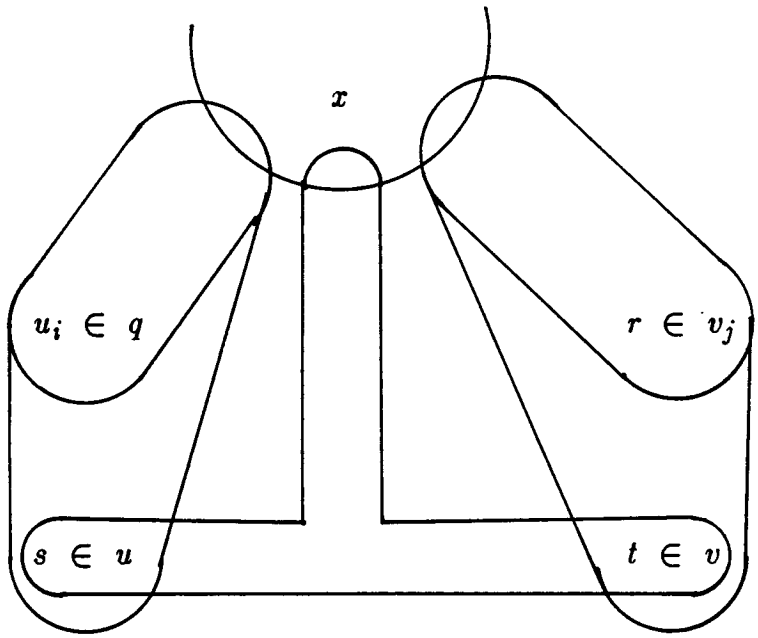

Figure 4-11: Proof of theorem 4-9, line 45, case 2

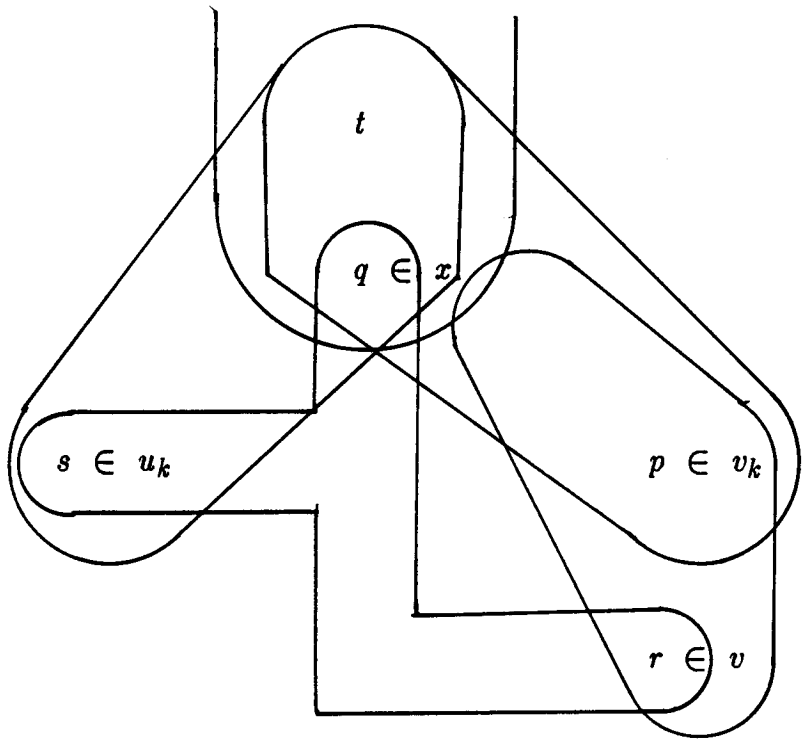

Figure 4-12: Proof of theorem 4-9, line 45, case 3

Lemma 4-10:

Assuming the hypergraph H has a directed path tree T, the following are always true at procedure *Visit* in algorithm *Find-PPT*:

1. P' is a PPT for $H[E']$ with root *Root*,
2. D is disjoint from V', and any vertex v adjacent in H to some vertex in D is a descendant in T of some vertex in *Root*,
3. All edges that have not yet been added to E' are either subsets of D or are disjoint from D.

Proof:

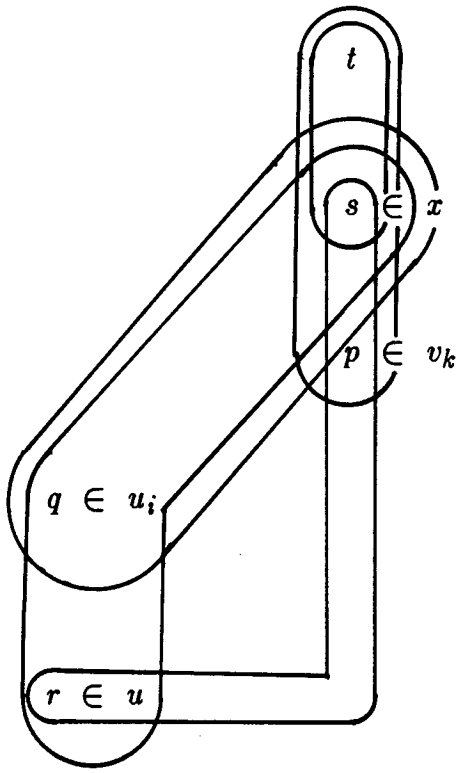

Figure 4-13: Proof of theorem 4-9, line 45, case 4

The lemma is clearly true initially, since D is empty and P' is the single vertex Root.

Let x be the vertex in P' being visited, c the edge in E-E' being added. By lemma 3-1 we know that $V' \cap c$ is a directed path in T (it is in $(H'+\{c\})^*$) so we can add it to P' first.

So assume, c is a subset of V'. There are several cases. Let L be the set of proper descendants of x that intersect c and have no proper descendants intersecting c. By lemmas 3-1, 3-7 and 3-10 for every $u \in L$

we can add the edge $(c \cap M(u)) \cup (M(v) \cap M(u))$ (where v is the parent of u in P') to E'. Procedure *Split* modifies P' by splitting u into $u \cap c$ and u-c, giving a PPT for the hypergraph obtained by adding these edges. For the rest of this proof we assume that this has been done and that each $u \in L$ is a subset of c.

We want to show that *Add-Edge* will either produce a PPT for $(V', E' \cup c)$ with the same root or it will produce a new root b such that the following conditions hold:

1. b intersects x.
2. The set of vertices V_b in P' that intersect b induces a subtree of P'.
3. All vertices in V_b, except possibly the root of the subtree induced by V_b, are subsets of b.
4. Let $Deep(V_b)$ be the set of vertices in V_b that have no proper descendants in V_b. All vertices in V' that are not descendants of some vertex in $Deep(V_b)$ are descendants in T are descendants in T of some vertex that is.
5. $M(v)$ is a subset of b for all v in $Deep(V_b)$.
6. b induces a directed path in any directed path tree for H.

A new root that satisfies these six conditions is called a *valid* root.

There are several cases to consider.

1. L is empty. Then P' is a PPT for $(V',E' \cup \{c\})$.

2. $L = \{u_m\}$, where $x,u_1, \ldots ,u_m$ is the directed (x,u_m) path in P'.

   a. If $u_1, \ldots ,u_m$ are subsets of c then P' is a PPT for $(V',E' \cup \{c\})$.

   b. Otherwise, suppose some u_i, $1 \leq i < m$, is not a subset of c. Let i be the minimum such value. Let $p \in u_m$, $q \in c \cap x$, $r \in u_i$-c. Using lemma 3-1 we can simplify $(V',E' \cup \{c\})$ to

   $$(\{p,q,r\},\{\{p,q,r\},\{p,q\},\{q,r\}\})$$

   and therefore conclude that all vertices in x are ancestors of p or of r in in T. Therefore, all vertices in $u_{i-1}, \ldots ,u_1$ and in x are ancestors in T of some vertex in u_m or in u_i, so the new root b=$M(u_i)$ is valid. See figure 4-14.

3. $L = \{u_m,v_n\}$, $x,u_1, \ldots ,u_m$ and $x,v_1, \ldots ,v_n$ directed paths in P', $v_k \neq u_k$, and k=1 or u_{k-1}=v_{k-1}. There are several cases:

   a. All u_i, v_j are subsets of c and no edge in E' intersecting u_k or v_k intersects *Above*(x,c). Using lemma 3-1 we can show that all vertices in proper ancestors of u_m and v_n are descendants in T of some vertex in u_m or v_n and, since $M(u_m)$ and $M(v_n)$ are subsets of c, the new root b=c is valid. See figure 4-15.

   b. All u_i, v_j are subsets of c, no edge in E' intersects both v_k and *Above*(x,c) but some edge in E' does intersect both u_k

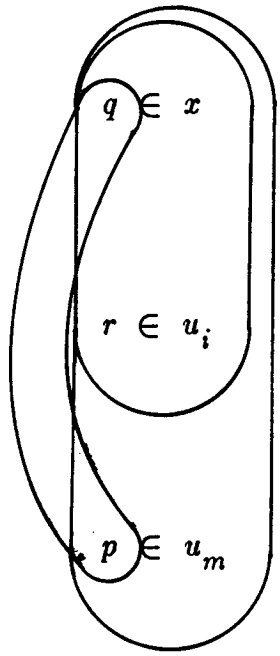

Figure 4-14: Proof of lemma 4-10, case 2b

and $Above(x,c)$. Let $p \in u_k$, $q \in x \cap c$, $r \in Above(x,c) \cap M(u_k)$, $s \in v_n$ and $t \in u_m$. We can simplify $(V',E' \cup \{c\})$ to

$$(\{p,q,r,s,t\},\{\{p,q,r\},\{p,q,t\},\{p,q,s,t\},\{q,r\},\{q,s\}\}).$$

This hypergraph has exactly one directed path tree, with t the root. By lemma 3-1 the vertex u_m contains a vertex that is an ancestor in T of all vertices in $M(u_m)$, so the new root $b = M(u_m)$ is valid. (The symmetric case where there is an edge in E' intersecting both v_k and $Above(x,c)$ is identical, with $b=M(v_n)$.) See figure 4-16.

c. A vertex u_i is not a subset of c, $k \leq i < m$. The algorithm will abort (correctly) if there is an edge intersecting v_k and *Above*(x,c) or if some v_j is not a subset of c. Let $p \in u_m$, $q \in x \cap c$, $r \in u_i$-c and $s \in v_n$. We can simplify $(V', E' \cup \{c\})$ to

$$(\{p,q,r,s\},\{\{p,q,r\},\{p,q,s\},\{q,r\},\{q,s\}\})$$

which has exactly one directed path tree, with root p. Therefore, all vertices in $M(u_m)$ are descendants in T of some vertex in u_m, and $b{=}M(u_m)$ is a valid root. See figure 4-17.

All other cases are ruled out by theorem 4-9.

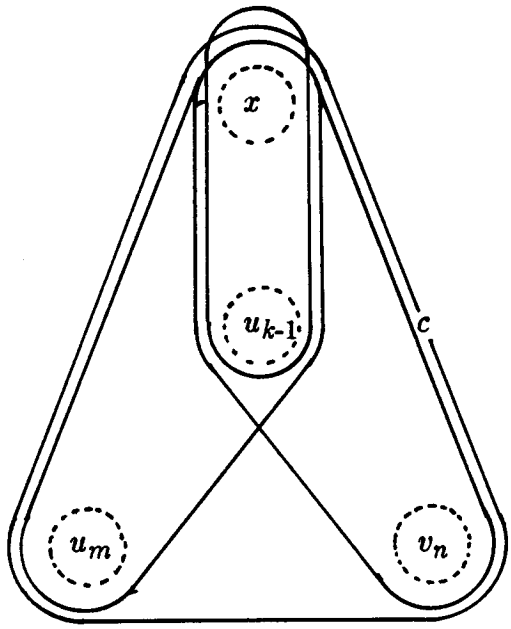

Figure 4-15: Proof of lemma 4-10, case 3a

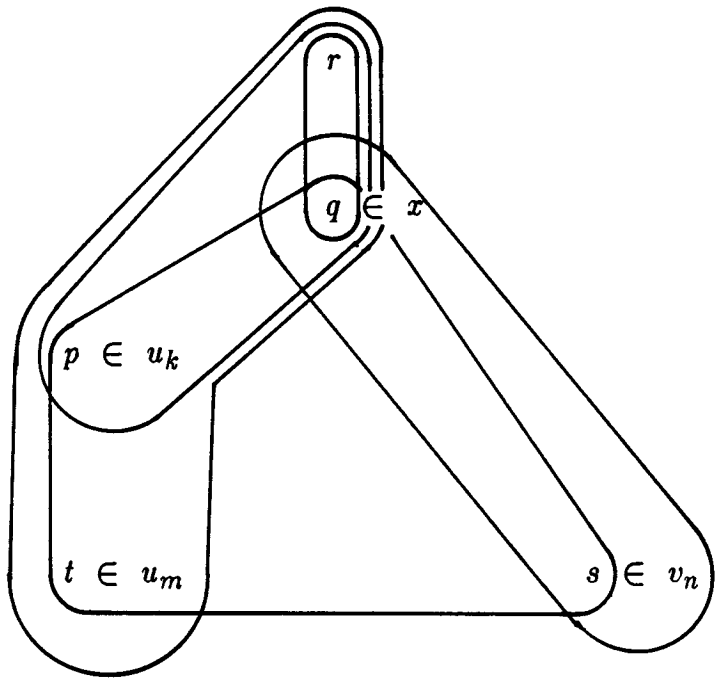

Figure 4-16: Proof of lemma 4-10, case 3b

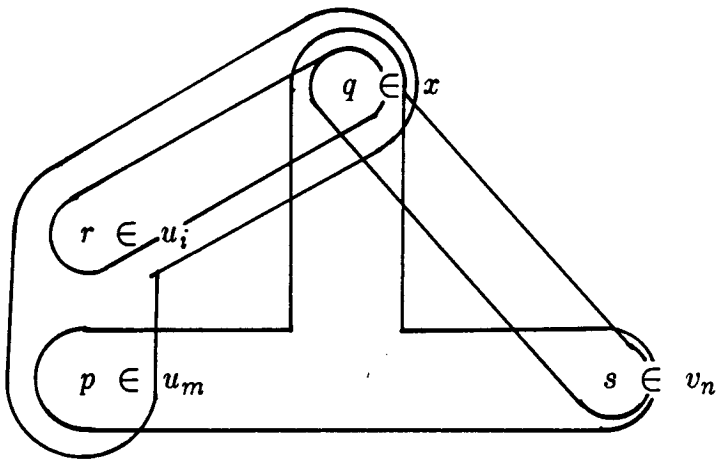

Figure 4-17: Proof of lemma 4-10, case 3c

Note that all the b's generated above are valid: (1) they intersect x

because the tree is built depth first and all edges intersecting descendants of x intersect x as well, (2) they all induce subtrees of P', (3) all vertices they intersect are subsets of b, except possibly the highest, (4) all vertices in b are descendants in T of some vertex in a vertex in $Deep(V_b)$, so (lemmas 3-1, 3-7) all vertices in V' not in descendants of some vertex in $Deep(V_b)$ are ancestors in T of some vertex in $\cup Deep(V_b)$, (5) $M(v)$ is a subset of b for v in $Deep(V_b)$ because either b *is* $M(v)$ or b is c and $M(u_i)$ and $M(v_i)$ are subsets of c for all i. Finally, (6) b induces a directed path in any directed path tree for H because it is either $c \in E$ or $M(u)$ for some vertex in P', and u does not intersect any edge intersecting D.

So, either P' is a PPT for $(V', E' \cup \{c\})$ (and we are done) or we have called *New-Root* on a valid root b. We now argue that if b is valid then the lemma remains true.

First, observe that *New-Root* discards (puts into D) only those vertices that are in V' and are not in b or in descendants of a vertex in $Deep(V_b)$, and all vertices that are disconnected from V' when these vertices are discarded. Since b is valid, all discarded vertices must be descendants in T of some vertex in $\cup Deep(V_b) \subseteq b$. Note also that no edge in E intersects both D and V-$(D \cup b)$, since $M(v)$ is a subset of b for all $v \in V_b$, and no hyperpath can go from a vertex in V'-b to a vertex in D without passing through a vertex in b (if not, H has a simple hypercircuit, contradicting the existence of a directed path tree T for H).

Therefore, after *New-Root* discards vertices, the union of the edges intersecting D is a subset of $D \cup b$. We need only verify that the resulting tree is a PPT, which is straightforward, and that all edges in E' appear in H^*, which is also straightforward. #

Putting it all together, we get

Theorem 4-11:

Find-PPT(V,E) either produces a PPT for (V,E) or correctly reports that (V,E) has no directed path tree.

Proof:

If *Find-PPT* aborts while building the first PPT then (theorem 4-9) (V,E) has no directed path tree. Otherwise, if it aborts after finding a root then (theorem 4-11, lemmas 4-8, 4-10) (V,E) has no directed path tree. If it finishes without aborting then (lemmas 4-3, 4-10) it produces a PPT for H. #

It remains to be shown that *Find-PPT* can be implemented to run in linear time. We use the same data structure as in *Find-PPT-with-Root*. A copy of the hypergraph $H=(V,E)$ is used to build the first PPT. H can be copied in linear time. When visiting a vertex x, sorting the edges incident on vertices in x takes time proportional to the sum of the sizes of the intersections (as in the first algorithm), or linear time overall. In *Add-Edge*, constructing the set A takes time

proportional to $|c|$. Finding whether A contains any vertices that are not descendants of x can be done in time proportional to the distance from x to the deepest vertex in A (if a nondescendant of x is actually in A it doesn't matter how much time we spend here, as long as it's linear in the size of H). The call to *Split* takes $O(|c|)$ time.

*Case*1 takes $O(|c|)$ time if a new root is not found, as does *Case*2. If a new root is found then *Case*1 and *Case*2 take time $O(|b|+|c|)$, where b is the new root, plus time proportional to the number of vertices discarded and to the number of vertex,edge pairs belonging to the discarded vertices. Since an edge is made the root at most once and vertices are discarded at most once, this will be linear time overall. The time spent here will cover the cost of determining if A contained any nondescendants of x (see above) because either all vertices on the directed path from x to some u in A are subsets of c (except for x), or they are all subsets the new root.

We conclude

Proposition 4-12:

Find-PPT can be implemented to run in time $O(|V|+SIZE(E))$.

4.3. Deriving Directed Path Trees from Partial Path Trees

We now give algorithms for producing directed path trees from partial path trees. The algorithms make use of a variant of PQ trees called PQR trees. We begin be describing PQR trees.

4.3.1. PQR Trees

A PQR tree is a tree that, like a PQ tree, represents a set of linear orderings of some set L. PQR trees are rooted, oriented trees. In addition to the leaves, P nodes and Q nodes of PQ trees, PQR trees have an additional type of internal node called *R nodes*. R nodes are drawn as parallelograms (see figure 4-18). No R node can be a child of another R node, and every R node must have at least two children. We will require that P nodes have at least three children (P nodes with two children are indistinguishable from Q nodes with the same children; these nodes will, by convention, be called Q nodes).[1]

Equivalence of PQR trees is the same as for PQ trees: two PQR trees are equivalent if one can be transformed into the other by permuting the order of children of some P nodes and reversing the order of the children of some Q nodes. The order of the children of R nodes cannot be changed. The definitions of *FRONTIER* and *CONSISTENT* also remain the same.

---

[1] Booth and Lueker [9] require that any node with exactly two children be a P node, but they note that this is entirely a matter of convention. The choice of Q node is more convenient here.

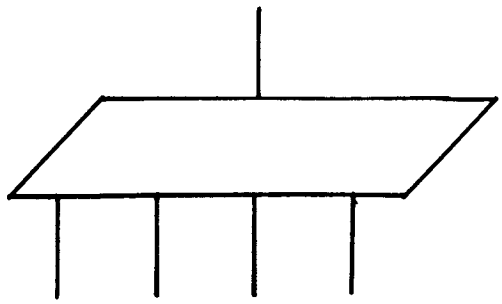

Figure 4-18: An R node

Theorem 4-13:

Let Z and Z' be PQR trees with $CONSISTENT(Z) = CONSISTENT(Z')$. Then, Z and Z' are equivalent.

Proof:

Clearly Z and Z' have the same set of leaves, otherwise they cannot be equivalent. Let Z and Z' each have k leaves. We proceed by induction on k.

If k=1 then each tree consists of a single node, so Z and Z' must be equivalent.

Otherwise, assume the theorem has been proved for trees with fewer than k leaves. Orient the trees so that they have the same frontier, and let x be the leftmost leaf. Let u and u' be the parent of x in Z and Z'. Obtain new PQR trees Y and Y' from Z and Z' by deleting x; the parent of x is removed if it has one child, and becomes a Q node if it previously was a P node with three children. By the induction hypothesis, Y and Y' are equivalent. The nodes u and u' are R nodes in Y and Y' iff u and u' are R nodes in Z and Z', so if u and u' are R nodes then Z and Z' are equivalent. If u and u' are both P nodes in Y and Y' they must also be P nodes in Z and Z', so Z and Z' are equivalent. If u and u' are Q nodes in Y and Y' but only one is a Q node in the larger tree then $CONSISTENT(Z) \neq CONSISTENT(Z')$: let x, y and z be the children of u (which we assume is a P node in Z). The frontier of u' (a Q node) in Y' is equal to the frontier of u in Y, so the frontier of u' in Z' is equal to the frontier of u in Z. u' has three children in Z', and is a Q node, so it cannot be transformed to have frontier x,$FRONTIER(z)$,$FRONTIER(y)$, as u can. This contradicts the assumption that $CONSISTENT(Z)$=$CONSISTENT(Z')$, so u and u' must either both be Q nodes or both be P nodes in Z and Z'. Therefore, Z and Z' are equivalent, completing the induction step. #

Define the *characteristic node* of an edge e in $E(H)$ in a PQ (PQR) tree Z, $CHAR(e)$, to be the deepest node in the tree such that every vertex in e is its

descendant. The set of edges in $E(H)$ with characteristic node t is $CHAR^{-1}(t)$. A node w in Z is said to be *empty with respect to e* if no vertex in $FRONTIER(w)$ is in A.

The following lemma is similar to lemma 1 in [35]:

Lemma 4-14:

Let Z be a PQR tree with leaves L, and let A be a subset of L that is consecutive in every sequence in $CONSISTENT(Z)$. Let t=$CHAR(A)$. Then,

1. t is a leaf, or
2. t is a P node and the nodes in $FRONTIER(t)$ are precisely those in A, or
3. t is a Q node or an R node and there is a consecutive sequence of children of t such that the set of nodes in the frontiers of these children is A.

Proof:

The proof is similar to the proof of lemma 1 in [35].

Assume t is not a leaf. If t is a P node, assume that there are vertices in $FRONTIER(t)$ not in A. Since t is the characteristic node of A it must have at least two children, say u and v, with descendants in A. If t has a child w empty with respect to e then permute the children of t so

that w occurs between u and v. The result is a PQR tree in which A is not consecutive, a contradiction. So, assume t has no child empty with respect to w. There must be some child of t, say x, with some leaf a in $FRONTIER(x)$ not in A. If a occurs first in $FRONTIER(x)$ then permute the children of t so that x occurs last. Otherwise, permute the children of t so that x occurs first. In either case A is not consecutive in $FRONTIER(t)$, again a contradiction.

If t is a Q node or an R node, let $u_i, \ldots, u_j$ be the children of t between the leftmost (u_i) and rightmost (u_j) children whose frontiers intersect A. Since t is the characteristic node of t, i is less than j. If any $FRONTIER(u_k)$ $(i < k < j)$ contains vertices not in A then A is not consecutive in $FRONTIER(t)$, a contradiction. Assume $FRONTIER(u_i)$ contains vertices not in A. These vertices must be the leftmost vertices in $FRONTIER(u_i)$. If t is a Q node then reverse the order of its children; in the resulting tree A is not consecutive in $FRONTIER(t)$. If t is an R node then u_i must be a P node or a Q node, so reverse the order u_i's children, also giving a tree in which A is not consecutive in $FRONTIER(t)$. So, $FRONTIER(u_i)$ is a subset of A. A symmetrical argument shows that $FRONTIER(u_j)$ is also a subset of A.

#

Using this lemma we can show

Lemma 4-15:

Let Z be a PQ tree for hypergraph (V,E), and let $A \subseteq V$. If $REDUCE(Z,A)$ is not the null tree then any node in Z empty with respect to A is also in $REDUCE(Z,A)$.

Proof:

Assume not. Let t be a node in Z with frontier F, F and A disjoint. We assume that the characteristic node of F in $REDUCE(Z,A)$ is not t but rather some other node t'. We can rule out the case where t and t' have the same vertices in their frontiers, since the order of the vertices in F in some sequence in $CONSISTENT(Z)$ cannot affect whether A is consecutive. So, $FRONTIER(t')$ must be a superset of F. By lemma 4-14 we know that t' cannot be a P node, so it must be a Q node. But this implies that the sequence obtained by reversing the order of the vertices in F in $FRONTIER(REDUCE(Z,A))$ is not in $CONSISTENT(REDUCE(Z,A))$, a contradiction. We conclude that the characteristic node of F in $REDUCE(Z,A)$ is t, proving the lemma. #

Let Z be a PQR tree and let a and b be disjoint nonempty sets of leaves in Z, where a, b and $a \cup b$ are consecutive in all sequences in $CONSISTENT(Z)$. The operation $Orient(a,b,Z)$ produces a new PQR tree Z' such that

$$CONSISTENT(Z') = \{\sigma \in CONSISTENT(Z) : a \text{ precedes } b \text{ in } \sigma\}.$$

Orient works as follows. If Z is the null tree then *Orient* returns it unchanged. Otherwise, let $t = CHAR(a \cup b)$. We know that $CHAR(a)$ and $CHAR(b)$ are either t or children of t. There are three cases depending on what kind of node t is.

1. t is a P node. This case cannot happen (see below).
2. t is a Q node. By lemma 4-14, there is a sequence of children of t $u_i, \ldots, u_k, u_{k+1}, \ldots, u_j$ such that the union of the frontiers of $u_i, \ldots, u_k$ is a and the union of the frontiers of $u_{k+1}, \ldots, u_j$ is b (or vice versa; reverse t in this case). Make t an R node. If t's parent is an R node then replace t with its children as children of t's parent. See figure 4-19.
3. t is an R node. Obtain the sequences of children as in the previous case. If a precedes b in this sequence then the tree is unchanged; otherwise, a null tree is produced.

The details of *Orient* are presented in figure 4-20.

Before we prove the correctness of *Orient*, a lemma is needed. This lemma will be used again later in this section.

Lemma 4-16:

Let Z be a PQR tree with leaves L, and let a,b, and $a \cup b$ be subsets of L consecutive in $FRONTIER(Z)$, a and b nonempty and disjoint. The node $CHAR(a \cup b)$ is a Q node or an R node, and each of the nodes $CHAR(a)$ and $CHAR(b)$ are either equal to or children of $CHAR(a \cup b)$.

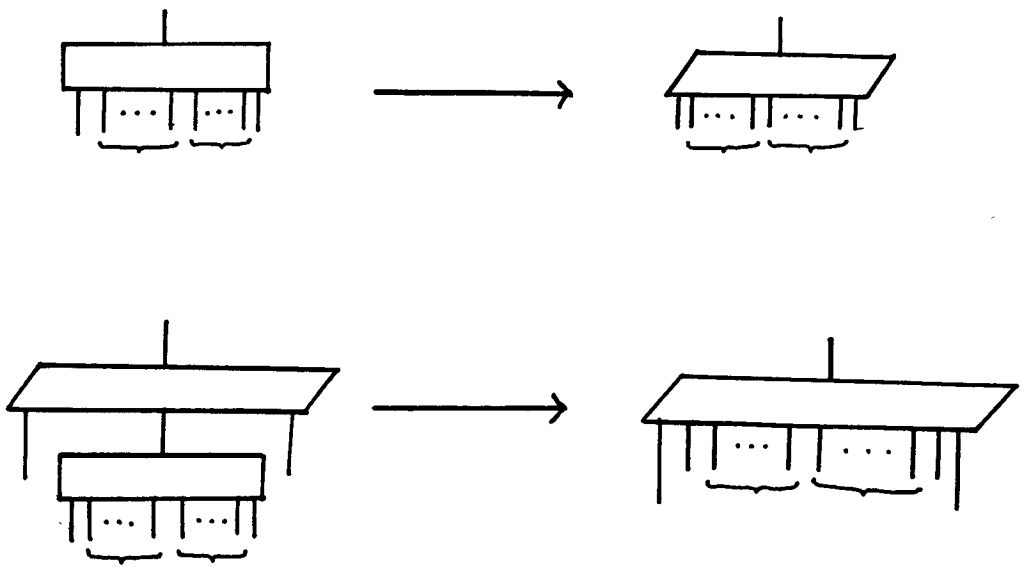

Figure 4-19: Action of *Orient* on a Q node

Proof:

Let $t = CHAR(a \cup b)$, $u = CHAR(a)$ and $v = CHAR(b)$. Clearly, u and v are descendants of t in Z. Assume u is not t or a child of t. There must be some node w, w a child of t and a proper ancestor of u in Z. Since b is consecutive in Z, w is empty with respect to b (otherwise, reverse the order of the children of w or of t; the result is a tree in which b is not consecutive in $FRONTIER(v)$). So, w must be an ancestor of some leave not in $a \cup b$. Again, reverse the order of the children of w or of t; $a \cup b$ is not consecutive in the resulting tree. So, u must be t or a child of t. The argument for v is the same.

Now assume that t is a P node, and therefore has at least three

Comment Make vertices in a precede those in b in Z;

proc *Orient*(a,b,Z)

begin

1. **if** $Z = \emptyset$ **then return**(Z);
2. $t := CHAR(a \cup b)$;
3. **if** t is a P node **then error**
4. **else if** t is a Q node **then**
5. **if** b precedes a in $FRONTIER(t)$ **then**
6. Reverse the order of the children of t;
7. Make t an R node;
8. **if** t is not the root and t's parent is an R node **then**
9. Replace t with t's children as children of t's parent
10. **else** /* t is an R node */
11. **if** b follows a in $FRONTIER(t)$ **then return**($\emptyset$);
12. **return**(Z)

end *Orient*;

Figure 4-20: Procedure *Orient*

children. By lemma 4-14, $FRONTIER(t) = a \cup b$, so one of a or b must

have vertices in the frontiers of at least two children of t. So, by lemma 4-14, $a \cup b = a$ or $a \cup b = b$, a contradiction. #

Lemma 4-17:

Orient is correct, and runs in $O(|a|+|b|)$ time.

Proof:

Let Z be a proper PQR tree and let a, b and $a \cup b$ be sets consecutive in $FRONTIER(Z)$, a and b nonempty and disjoint. If Z is the null tree then the lemma is true. Otherwise, assume Z is not the null tree. Let $t = CHAR(a \cup b)$, $u = CHAR(a)$ and $v = CHAR(b)$. By lemma 4-16 u and v are equal to or children of t, and t is a Q node or an R node. In either case a and b are unions of the frontiers of consecutive children of t (lemma 4-14). (These children can be found in $O(|a|+|b|)$ time.) If t is an R node then either the children of t with frontiers intersecting a precede those whose frontiers intersect b, so all in all sequences in $CONSISTENT(Z)$ a precedes b, or the children with frontiers intersecting a follow those that intersect b, in which case in no sequence in $CONSISTENT(Z)$ does a precede b, so *Orient* is correct.

If t is a Q node then any sequence in $CONSISTENT(Z)$ in which a precedes b may be obtained by orienting t so that a precedes b then performing operations on other nodes in the tree. A PQR tree allowing precisely these operations is obtained by permuting t so that a precedes b

then fixing it to be an R node. So, the PQR tree produced by *Orient* is consistent with exactly those sequences in $CONSISTENT(Z)$ in which a precedes b.

The running time of *Orient* is simply the time needed to find t, u and v, plus the time to manipulate t (which can be done in $O(1)$ time), for a total of $O(|a|+|b|)$ time. #

4.3.2. Using PQR Trees to Obtain Directed Path Trees

We now describe the algorithm that uses PQR trees to obtain directed path trees for H from a partial path tree for H. The algorithm makes use of Booth and Lueker's procedure *Reduce* as well as *Orient*, described in the previous subsection.

For the rest of this subsection, let H be a hypergraph, P a PPT for H, and let x be a vertex in P.

Definition 4-18:

Let T be a directed path tree for the hypergraph H. We say a connector $CONN(x)$ is *oriented inwards* in T if the vertices in $d(x,i)$ are ancestors in T of those in $d(x,i+1)$, $(1 \leq i < |CONN(x)|)$; otherwise, $CONN(x)$ is *oriented outwards*. See figure 4-21.

We know (lemma 3-9) that if $|CONN(x)| > 1$ then $CONN(x)$ must be oriented inwards or outwards. The notion of orientation is not defined for $|CONN(x)| \leq 1$ (such connectors are called *trivial*).

Figure 4-21: Inwards, Outwards Orientations

Lemma 4-19:

Let T be a directed path tree for H. If $|CONN(x)| > 1$ then the root of T is in $Below(x)$ iff $CONN(x)$ is oriented inwards.

Proof:

($\Leftarrow$) The vertices in $c(x,1)$ are ancestors in T of the vertices in $e(x,2)$ (lemma 3-8). By lemmas 3-3 and 3-11 all vertices in $V(H)$-$Below(x)$ are descendants in T of some vertex in $c(x,1)$, which must be a descendant of some vertex in x. Therefore, the root of T is in $Below(x)$.

($\Rightarrow$) Assume the root of T is in $Below(x)$. If $CONN(x)$ is oriented

outwards then (lemma 3-8) all vertices in $e(x,2)$ are ancestors in T of all vertices in x. This contradicts the assumption that the root of T is in $Below(x)$, since we can trace a simple hyperpath in H^* from each vertex in x to each vertex in $Below(x)$ (simply add edges of the form $m(v)$, v a vertex in P). #

We use PPT's to decompose P into simple pieces. The next few lemmas show how to do this so the directed path trees for the pieces may be fitted together again.

Lemma 4-20:

Let T be a directed path tree for H. If x is a vertex in PPT P other than the root then either the root of $T[Below(x)]$ is in x or the root of $H\text{-}Below(x)$ is in $m(x)\text{-}x$.

Proof:

If the root of T is in $V(H)\text{-}Below(x)$ then, by lemmas 3-11 and 3-3, the vertices in x are ancestors in T of all vertices in $Below(x)\text{-}x$, so the root of $T[Below(x)]$ is in x.

Otherwise, assume the root of T is in $Below(x)$. Again, by lemmas 3-3 and 3-11 we can show that all vertices in $V\text{-}(Below(x) \cup m(x))$ are ancestors in T all vertices in $m(x)$, so the root of $T\text{-}Below(x)$ must be in $m(x)\text{-}x$. #

We can prove a restricted version of the converse of lemma 4-20.

Lemma 4-21:

If $|CONN(x)|=1$ then for any directed path trees T_1 for $H[Below(x)]$ and T_2 for H-$Below(x)$, of the root of T_1 is in x then there is a directed path tree T for H that simplifies to T_1 and T_2.

Proof:

T is obtained by making the roots of T_1 a child of the deepest vertex in $m(x)$-x in T_2 (by lemmas 3-1 and 3-7, $m(x)$-x must induce a directed path in T_2). For any edge c in $E(H)$ intersecting both $Below(x)$ and $V(H)$-$Below(x)$, the intersection of c and $V(H)$-$Below(x)$ must be $m(x)$-x (since $|CONN(x)|=1$), so c is a path in T. Since T_1 and T_2 are subtrees of T all other edges also induce paths in T, so T is a directed path tree for H. #

Lemma 4-22 gives necessary and sufficient conditions for a hypergraph H to have a directed path tree whose root is in the root of a partial path tree P.

Lemma 4-22:

Let x be the root of PPT P. The hypergraph H has a directed path tree with root in x iff (1) for each child y of x, $P[Below(y)]$ has a directed path tree with root in y, and (2) the hypergraph

$$H^x = H[x] + \{e(y,2) : y \text{ a child of } x, |CONN(y)| > 1\} \quad (1)$$

has a directed path tree in which for each child y of x, $|CONN(y)| > 1$, the vertices in $e(y,2)$ are ancestors of those in $c(y,1)$.

Proof:

($\Rightarrow$) Let T be a directed path tree for H with root in x. The set x induces a directed path in T, and each $Below(y)$ induces a subtree of T rooted at some child in T of a vertex on this path. Deleting the vertices in $V(H)$-$Below(y)$ yields the desired directed path tree for $P[Below(y)]$. In addition, in the directed path tree $T[x]$ each $e(y,2)$ must induce a directed path (lemma 3-8) and all vertices in $e(y,2)$ must precede those in $c(y,1)$ (otherwise, $c(y,1) \cup y$ does not induce a path in T), so $T[x]$ is the desired directed path tree for $H[x]$.

($\Leftarrow$) Let T^x be a directed path tree for H^x in which for each child y of x where $|CONN(y)| > 1$, the vertices in $e(y,2)$ are ancestors of those in $c(y,1)$, and for each child y of x let T^y be a directed path tree for $H[Below(y)]$ with root in y. Build the directed path tree for H by making the root of T^y a child of the deepest vertex in $e(y,1)$ in T^x. Any

edge in $E(H)$ entirely in T^x or one of the T^y's induces a directed path in this tree. Let c be some edge in $E(H)$ intersecting both x and y for some child y of x. Since the last vertex in $e(y,1)$ in T^x is in $c(y,1)$, the last vertex in $e(y,1)$ is also the last vertex in T^x of the directed path $c \cap x$, so c induces a directed path in T. #

Figure 4-22 gives the algorithm based on lemma 4-22. It returns $\emptyset$ if $H=(V,E)$ has no directed path tree with root in x.

Theorem 4-23:

The procedure *Find-Path-Tree-with-Root* returns a directed path tree for H with root in $root(T)$ if one exists, otherwise, it returns $\emptyset$.

Proof:

Immediate from lemma 4-22 and the correctness of *Orient* (lemma 4-17). #

The general algorithm must determine where the root of the directed path tree can be. The next few lemmas will help determine where the root can be.

Comment Find a path tree for $H\,[Below(x)]$ with root in x, x the root of P;

proc *Find-Path-Tree-with-Root* (P)

begin

1. $x := root\,(P)$;
2. **for** each child y of x **do**
3. $\quad P^y :=$ the subtree of P rooted at y;
4. $\quad T^y := Find\text{-}Path\text{-}Tree\text{-}with\text{-}Root\,(P^y)$;
5. $\quad$ **if** $T^y = \emptyset$ **then return**($\emptyset$)
6. $Z :=$ a PQ tree for P^x;
7. **for** each child y of x **do**
8. $\quad$ **if** $|CONN(y)| > 1$ **then**
9. $\quad\quad Z := Orient(Z,e(y,2),c(y,1))$;
10. $\quad\quad$ **if** $Z = \emptyset$ **then return**($\emptyset$)
11. $T :=$ a path consisting of the vertices in $FRONTIER\,(Z)$, in that order;
12. **for** each child y of x **do**
13. $\quad$ Make the root of T^y a child of the deepest vertex in $c(y,1)$ in T;
14. **return**(T)

end *Find-Path-Tree-with-Root*

Figure 4-22: Procedure Find-Path-Tree-with-Root

Lemma 4-24:

Let T be a directed path tree for H, and let x, y and z be vertices in PPT P, x the parent of y, z the parent of x. If the root of T is in $Below(x)$, $CONN(x)$ and $CONN(y)$ are each nontrivial, and $M(y)$ intersects z then the root of T is in $Below(y)$.

Proof:

Since $|CONN(x)| > 1$, x induces a directed path in T and $CONN(x)$ is oriented inwards (lemmas 3-8 and 4-19). Let k=$|CONN(y)|$; we know that, for some i, $c(x,i)$ is a subset of $c(y,k)$ (because $M(y)$ intersects z), so the vertices in $c(y,k\text{-}1)$ are ancestors of those in $c(y,k)$ in T (lemma 3-8) and $CONN(y)$ is oriented inwards in T. Therefore, the root of T is in $Below(y)$ (lemma 4-19). #

Lemma 4-25:

Let $|CONN(x)|$=1, and suppose every directed path tree T for H has its root in $Below(x)$-x. Then, H has a directed path tree iff there exists a directed path tree T_1 for H-$Below(x)$ with root in $m(x)$-x and a directed path tree T_2 for $H[Below(x) \cup m(x)]$. Moreover, for any two such T_1 and T_2 there is a directed path tree T for H such that T-$Below(x) = T_1$ and $T[Below(x) \cup m(x)]_{m(x)\text{-}x} = (T_2)_{m(x)\text{-}x}$.

Proof:

($\Rightarrow$) If H has a directed path tree T then (lemmas 3-1, 4-20) the root of

tree *T-Below*(x) (which is a directed path tree for *H-Below*(x)) in in $m(x)$-x, and the tree $T[Below(x) \cup m(x)]$ is a directed path tree for $H[Below(x) \cup m(x)]$.

($\Leftarrow$) Let T_1 and T_2 be directed path trees for *H-Below*(x) and $H[Below(x) \cup m(x)]$, $root(T_1)$ in $m(x)$-x. Build T by replacing the directed path in T_2 induced by $m(x)$-x with the path induced by $m(x)$-x in T_1 (along with the rest of T_1). Since T_1 is a subtree of T, every edge in $E(H)$ that is a subset of $V(H)$-*Below*(x) induces a directed path in T, as does every edge that is a subset of *Below*(x). Since $|CONN(x)|=1$, $m(x)$-x is a subset of every edge that intersects both x and $m(x)$-x, so each such edge induces a path in T. So, every edge in $E(H)$ induces a path in T and T is a directed path tree for H. #

Lemma 4-26:

Let H^x be as in equation 1 (lemma 4-22, and let y be a child of x, $|CONN(y)| > 1$. The characteristic node of $e(y,1)$ in a PQ tree Z for H^x is a Q node.

Proof:

Since the disjoint sets $e(y,2)$ and $c(y,1)$ are consecutive in Z, by lemma 4-16 the characteristic node of $e(y,1)=e(y,2) \cup c(y,1)$ is either a Q node or an R node. Since Z is a PQ tree is must be a Q node. #

Let $CHAR(e(y,1))$ be the characteristic node of $e(y,1)$ in a PQ tree Z for H^x. The connector $CONN(y)$ is oriented inwards in Z if $c(y,1)$ precedes $e(y,2)$ in $FRONTIER(Z)$, otherwise, it is oriented outwards.

We now outline the full algorithm for making a directed path tree given a partial path tree. Let $H=(V,E)$ be a hypergraph, P a PPT for H.

1. Use procedure *Find-Path-Tree-with-Root* to decide for each vertex x in P whether $H[Below(x)]$ has a path tree with root in x, and to produce such a tree if one exists (theorem 4-23). If the root has such a tree then stop; we are done. If there are two unrelated vertices x and y such that $H[Below(x)]$ has no path tree with root in x and $H[Below(y)]$ has no path tree with root in y then stop; H has no path tree (this follows from lemma 4-20). Otherwise, the vertices for which $H[Below(x)]$ has no path tree with root in x form a path from the root to some vertex x.
2. Prune from P all subtrees rooted at each vertex y such that $H[Below(y)]$ has a directed path tree with root in y and $|CONN(y)|=1$. By lemma 4-21 the directed path trees we've found for the $H[Below(y)]$ may be connected to any directed path tree for the rest of the hypergraph. Remove from H all vertices in vertices pruned from P. P remains a PPT for H.
3.
    a. If $|CONN(x)|=1$ then H may be decomposed into the hypergraphs $H\text{-}Below(x)$ and $H[Below(x) \cup m(x)]$. By lemma 4-20 $H\text{-}Below(x)$ must have a directed path tree with root in $m(x)$-x; create a PPT for

H-Below(x) with root $m(x)$-x (use Find-PPT-with-Root) and use Find-Path-Tree-with-Root to find the directed path tree with root in $m(x)$-x. A PPT for $H[Below(x) \cup m(x)]$ can be produced by adding the vertices in $m(x)$ to x and deleting all vertices that are not descendants of x. By lemma 4-25, the directed path tree T_1 for *H-Below*(x) with root in $m(x)$-x and any path tree T_2 for $H[Below(x) \cup m(x)]$ may be combined (replace $m(x)$-x in T_2 with the path $m(x)$-x in T_1) to get a directed path tree for H. Use case 3.b.ii. below to find the path tree for $H[Below(x) \cup m(x)]$.

b. Otherwise, if $|CONN(x)| \neq 1$ there are two subcases:

   i. The root of P is not x, and there is a child y of x such that $M(y)$ intersects the parent of x. In this case lemma 4-24 guarantees that the root of any directed path tree for H is in *Below*(y), so repeatedly set x equal to y until x no longer has a child y with $M(y)$-$(x \cup y)$ nonempty. Since *Below*(y) has a path tree with root in y we know that $|CONN(y)| > 1$, otherwise y would have been pruned off at step 2. If two such y's are found then stop; H has no directed path tree (lemma 4-24). When done, we have an x such that every directed path tree for H has root in *Below*(x), $H[Below(x)]$ has a directed path tree T_1 with root in x and for every child y of x no edge in $E(H)$ intersects both y and the parent of x. Build a path tree for H by finding a directed path tree T_2 for *H-Below*(x) with root in $m(x)$-x (use Find-PPT-with-

Root and Find-Path-Tree-with-Root); make the root of T_2 a child of the deepest vertex in x in T_1.

ii. The root of P is x or there is no child y of x such that $M(y)$ intersects the parent of x. By lemma 4-22, it must be the case that $H[x]$ has no directed path tree in which all $CONN(y)$ (y a child of x) are oriented outwards. By lemma 4-26, the characteristic nodes of the $c(y,1)$ are Q nodes in a PQ tree Z for P^x. The only way we can fail to be able to orient all connectors outwards is if two connectors, say $CONN(u)$ and $CONN(v)$ ($|CONN(u)|,|CONN(v)| > 1$) have the same characteristic Q node and are oriented in opposite directions. Call such a Q node *conflicting.* There are several cases:

1. There are at least two conflicting Q nodes. This requires that at least two connectors be oriented inwards in any directed path tree, an impossibility (lemma 4-19), so stop; H has no directed path tree.
2. There is a conflicting Q node with at least two Q nodes oriented in each direction. Again, at least two connectors must be oriented inwards, so H has no directed path tree (lemma 4-19).
3. Otherwise, there is a single conflicting Q node. It has one connector, say $CONN(y)$, oriented in one direction. In the other direction it either has more than one connector or it

has a single connector; for the moment assume the former. In this case the root of any directed path tree for H must be in $Below(y)$ (lemma 4-19). We can now repeat the process used in case 3.b.i.: while y has any child z such that $M(z)$ intersects y's parent, we know that the root of any path tree for H is in Below(z), so let y be z. If two such z's are found then stop; H has no directed path tree. Eventually, a vertex y is found such that $H[Below(y)]$ has a directed path tree with root in y and no child z of y is in an edge intersecting y's parent. As in case 3.b.i., build a directed path tree for H by building a directed path tree for H-$Below(y)$ with root in $m(y)$-y, and make the root of this tree a child of the deepest vertex in y in the directed path tree for $H[Below(y)]$.

If the conflicting Q node had one connector oriented in each direction, say $CONN(u)$ and $CONN(v)$, we try the procedure outlined above for $y=u$ and, if that fails, for $y=v$. H has a directed path tree iff it has a directed path tree with root in $H[Below(u)]$ or in $H[Below(v)]$.

Theorem 4-27:

This algorithm is correct and runs in linear time.

Proof:

The proof of correctness is straightforward (see comments above for details).

The running time of the algorithm of each part of the algorithm is $O(|V|+SIZE(E))$ so the algorithm runs in linear time. #

4.4. Summary of Results for Chapter 4

In this chapter we have presented several algorithms and have proved them correct. The first algorithm, *Find-PPT-with-Root*, determines if a hypergraph H has a partial path tree with a given root. It was described in section 4.1 and runs in linear time. The second algorithm, *Find-PPT*, will produce a partial path tree for a directed path hypergraph in linear time (on hypergraphs that are not directed path hypergraphs it may abort). It was described in section 4.2. In section 4.3 we defined PQR trees and showed how to use them to obtain directed path trees from partial path trees. This procedure also takes linear time.

5. An Isomorphism Algorithm for Directed Path Graphs

This chapter gives a polynomial time algorithm for deciding if two directed path hypergraphs H and H' are isomorphic. Since two directed path graphs are isomorphic iff the duals of their clique hypergraphs are isomorphic, the algorithm also enables one to decide quickly if two directed path graphs are isomorphic (using the fact that the clique hypergraph of a chordal graph can be found in linear time with the Rose-Tarjan-Lueker algorithm). The algorithms presented here use a variant of Lueker and Booth's PQ tree isomorphism algorithm [35] [14], which is in turn based on the the classic linear time tree isomorphism algorithm [12] [1].

5.1. Edge Labelled PQR Tree Isomorphism

Definition 5-1:

Two PQ (PQR) trees Z and Z' are *isomorphic* if there is a bijection θ from the nodes of Z to the nodes of Z' such that the image of Z under θ is equivalent to Z'.

Let A and A' be families of subsets of $FRONTIER(Z)$ and $FRONTIER(Z')$. Let lbl and lbl' be functions from A and A' to some set of labels. The pairs (Z,A) and (Z',A') are *edge labelled isomorphic* iff there is an isomorphism θ from Z to Z' and a bijection ϕ from A to A' such that for every set a in A, $\phi(a)=\{\theta(v):v \in a\}$ and $lbl(a)= lbl'(\phi(a))$.

Theorem 5-2:

Let H and H' be interval hypergraphs, and let Z and Z' be PQ trees for H and H'. Let *lbl* and *lbl'* be constant valued functions, and let A and A' be the families $E(H)$ and $E(H')$. The pairs (Z,A) and (Z',A') are edge labelled isomorphic iff H and H' are isomorphic.

Proof:

($\Rightarrow$) Any edge labelled isomorphism from (Z,A) to (Z',A') induces an isomorphism from the hypergraph $(FRONTIER(Z),A)$ to the hypergraph $(FRONTIER(Z'),A')$; i.e., from H to H'.

($\Leftarrow$) If H and H' are isomorphic then for any PQ trees Z and Z' for H and H', there is an bijection θ from the leaves of Z to the leaves of Z' such that $CONSISTENT(\theta(Z))=CONSISTENT(Z')$. So, by theorem 2-11 (and theorem 4-13) $\theta(Z)$ and Z' are equivalent, so Z and Z' are isomorphic. The bijection on the leaves of Z comes from the isomorphism from H to H', so it also maps the edges of H to the edges of H'. #

Figure 5-1 gives an algorithm for labelling the nodes of edge labelled PQR trees. If Z is a PQR tree and A a family of subsets of $FRONTIER(Z)$, where each a in A is consecutive in every σ in $CONSISTENT(Z)$, and u is a node in Z, then let Z_u be the subtree of Z rooted at u and let A_u be the subfamily of A consisting of those sets in A that are subsets of $FRONTIER(u)$. If u and v are nodes in Z,

depth(u)=*depth*(v), then the labelling algorithm will make the labels $L[u]$ and $L[v]$ the same iff the edge labelled PQR trees (Z_u,A_u) and (Z_v,A_v) are isomorphic.

In figure 5-1, let *SORTED*(x_1,...,x_n) denote the list

"(" || $x_{\pi(1)}$ || "," || . . . || "," || $x_{\pi(n)}$ || ")"

where || denotes concatenation, the x_i are integers or strings (the strings may contain commas and parentheses if the parentheses are balanced and all commas appear inside a set of parentheses), π is a permutation on $\{1, \ldots, n\}$ and $x_{\pi(i)} \leq x_{\pi(i+1)}$, $1 \leq i < n$ (where the comparison is lexicographic if the x_j's are strings). Observe that if A and A' are families of integers or families of strings then $SORTED(A) = SORTED(A')$ iff there is a bijection f from A to A' such that $f(a)=a$ for all a in A.

Comment Algorithm to label the nodes of an edge-labelled PQR tree;

proc *PQRLabel(Z,A,lbl)*

begin

1. **for** i := 0 **to** $depth(Z)$ **do** $D[i]$:= $\{ v \in V(Z) : depth(v)=i \}$;
2. **for** i := $depth(Z)$ **downto** 0 **do**
3. **for** all v in $D[i]$ **do**
4. Let $v_1, \ldots, v_k$ be the children of v, and
5. let $a_1, \ldots, a_m \in CHAR^{-1}(v)$;
6. **if** v is a leaf **then**
7. $EL[v]$:= $SORTED(lbl(a_1), \ldots, lbl(a_m))$;
8. $SL[v], kind$:= ϵ, "L";
9. **else if** v is a P node **then**
10. $EL[v]$:= $SORTED(lbl(a_1), \ldots, lbl(a_m))$;
11. $SL[v]$:= $SORTED(I[v_1], \ldots, I[v_k])$;
12. Permute the children of v so that their I numbers occur in
13. nondecreasing order from left to right;
14. $kind$:= "P"
15. **else** /* v is a Q node or an R node */
16. **for** j := 1 **to** m **do**
17. b_i := "(" || i_1 || "," || i_2 || "," || $lbl(a_j)$ || ")"
18. and c_i := "(" || i_2 || "," || i_1 || "," || $lbl(a_j)$ || ")"

Figure 5-1: Procedure *PQRLabel* (part 1)

19. where $1 \leq i_1 < i_2 \leq k$ and $a_j = \cup_{r=i_1}^{i_2} FRONTIER(v_r)$;
20. **if** v is an R node **then**
21. $SL[v]$:= "(" || $I[v_1]$ || "," ... "," || $I[v_k]$ || ")";
22. $EL[v]$:= $SORTED(b_1, \ldots, b_m)$;
23. $kind$:= "R"
24. **else** /* v is a Q node */
25. $S1$:= "(" || $I[v_1]$ || "," ... "," || $I[v_k]$ || ")";
26. $S2$:= "(" || $I[v_k]$ || "," ... "," || $I[v_1]$ || ")";
27. $E1$:= $SORTED(b_1, \ldots, b_m)$;
28. $E2$:= $SORTED(c_1, \ldots, c_m)$;
29. **if** $S1 < S2$ or ($S1 = S2$ and $E1 \leq E2$) **then**
30. $SL[v]$:= $S1$;
31. $EL[v]$:= $E1$
32. **else**
33. Reverse the order of the children of v;
34. $SL[v]$:= $S2$;
35. $EL[v]$:= $E2$
36. $kind$:= "Q";
37. $L[v]$:= $kind$ || $EL[v]$ || $SL[v]$;
38. **for** all v in $D[i]$ **do**
39. $I[v]$:= the index of $L[v]$ in the set $\{L[u]: u \in D[i]\}$

end *PQRLabel*;

Figure 5-1: Procedure *PQRLabel* (part 2)

Theorem 5-3:

Let u and v be nodes in Z at the same depth. *PQRLabel* makes $L[u]=L[v]$ (and $I[u]=I[v]$) iff there is an edge-labelled isomorphism from (Z_u,A_u) to (Z_v,A_v).

Proof:

The theorem is proved by induction on $h=depth(Z)\text{-}depth(u)$. Assume the theorem has been proved for all deeper vertices. Let the nodes u and v have children $u_1, \ldots, u_k$ and $v_1, \ldots, v_l$, respectively. We want to show that $L[u]=L[v]$ iff there is an edge labelled isomorphism from (Z_u,A_u) to (Z_v,A_v).

($\Rightarrow$) Suppose the labels $L[u]$ and $L[v]$ are equal. Then, u and v are the same kind of node (P, Q, R or leaf), $EL[u]=EL[v]$ and $SL[u]=SL[v]$.

Assume u and v are leaves or P nodes. Since $SL[u]=SL[v]$, there is a bijection from the children of u to the children of v (say, from u_i to v_i) such that (Z_{u_i},A_{u_i}) is edge-labelled isomorphic to (Z_{v_i},A_{v_i}). This implies that u and v have the same number of children, of course. All elements a in A_u are either in every A_{u_i} (and therefore in $CHAR^{-1}(u)$), or in exactly one A_{u_i} (and not in $CHAR^{-1}(u)$). Since $EL[u]=EL[v]$, there is a bijection from $CHAR^{-1}(u)$ to $CHAR^{-1}(v)$ preserving edge labels. Therefore, there is an edge labelled isomorphism from (Z_u,A_u) to (Z_v,A_v).

If u and v are R nodes then since $SL[u]=SL[v]$, we know that $k=l$ and there are edge labelled isomorphisms between the subtrees rooted at the u_i and the v_i as in the previous case. In addition, since $EL[u]=EL[v]$, there is a bijection from $CHAR^{-1}(u)$ to $CHAR^{-1}(v)$ mapping edges $a=$ $FRONTIER(u_i) \cup \ldots \cup FRONTIER(u_j)$ to an edge containing vertices $FRONTIER(v_i) \cup \ldots \cup FRONTIER(v_j)$, the image edge having the same label. Therefore, there is a edge labelled isomorphism from (Z_u,A_u) to (Z_v,A_v). A similar argument applies for Q nodes.

($\Leftarrow$) Assume there is an edge labelled isomorphism (θ,ϕ) from (Z_u,A_u) to (Z_v,A_v), u and v distinct nodes at depth $depth(Z)$-h. If u and v are leaves this implies that there is an bijection from A_u to A_v preserving edge labels, so $EL[u]=EL[v]$ and therefore $L[u]=L[v]$. Otherwise, if u and v are not leaves then arrange the PQR tree so that the isomorphism maps the i-th child of u, u_i, to the i-th child of v, v_i. Restricting (θ,ϕ) to Z_{u_i} and Z_{v_i} yields an edge labelled isomorphism, so by the induction hypothesis $L[u_i]=L[v_i]$.

If u and v are P nodes then the function ϕ restricted to $CHAR^{-1}(u)$ and $CHAR^{-1}(v)$ is a bijection preserving edge labels, so $EL[u]=EL[v]$. Since $L[u_i]=L[v_i]$, we know $SL[u]=SL[v]$, and so $L[u]=L[v]$.

If u and v are R nodes then ϕ restricted to $CHAR^{-1}(u)$ and $CHAR^{-1}(v)$ is a bijection mapping edges

$$a_i = \boldsymbol{FRONTIER}(u_{i_1}) \cup \ldots \cup \boldsymbol{FRONTIER}(u_{i_2})$$

(where $1 \leq i_1 < i_2 \leq k$) to

$$\phi(a_i) = FRONTIER(v_{i_1}) \cup \ldots \cup FRONTIER(v_{i_2}).$$

So, $EL[u]=EL[v]$. Since $L[u_i]=L[v_i]$ for $i=1, \ldots, k$, we know $SL[u]=SL[v]$, and so $L[u]=L[v]$.

Finally, if u and v are Q nodes then the labels for u and v produced by *PQRLabel* can be obtained by orienting u and v so that the isomorphism θ maps u_i to v_i, $i=1, \ldots, k$, and then reversing u and v if $S1 > S2$, or $S1=S2$ and $E1 > E2$. Because (Z_u,A_u) and (Z_v,A_v) are isomorphic, *PQRLabel* will reverse u iff it reverses v. Therefore, by arguments similar to those for R nodes, $L[u]=L[v]$. #

The procedure *PQRLabel* can be implemented to run in time proportional to the number of nodes in Z plus the sums of the sizes of the elements of A plus the sizes of their labels. We can find the characteristic nodes of the elements of A in linear time. The sorting steps of the algorithm can be implemented to run in linear time through standard techniques (see [1], [35], [14]).

5.2. Partial Path Tree Respecting Isomorphisms

Let H and H' be directed path hypergraphs, and let P and P' be PPT's for H and H'. For any subset U of $V(H)$ and any edge $e=E(H[U])(i)$ (where $i \in I(E(H[U]))$), the *degree of e in H* is $deg_H(e)=|E(H)(i)|$ (similarly for $e'=E(H'[U'])(i)$).

An isomorphism (θ,ϕ) from $H[U]$ to $H'[U']$ *respects P and P'* if (1) for any vertices x and y in U, $P(x)$ is a descendant of $P(y)$ iff $P'(\theta(x))$ is a descendant of $P'(\theta(y))$, and (2) for any edge e in $E(H[U])$, $deg_H(e)=deg_{H'}(\phi(e))$. Theorem 3-5 implies that any isomorphism from H to H' that maps $root(P)$ to $root(P')$ respects P and P'.

Our problem, then, is to determine if there is an isomorphism from H to H' that respects P and P'. This will be done by another variant of the tree isomorphism algorithm. This algorithm works by finding labelling each vertex in P and P' with a canonical representation for the hypergraph $H[Below(x)]$ (or $H'[Below(y)]$). The canonical representations are found by computing the canonical representation for the child of the vertex (in the PPT), then using the indices of these canonical representations as labels to label the PQR tree for the interval hypergraph H^x (or, H'^y). The computation of canonical representations is done by procedure *Canon*. The canonical representations of the labelled PQR trees is done with *PQRLabel*, presented above, and the procedure *Trav*, which traverses the PQR tree produced by *PQRLabel*, converting it to a parenthesized string.

Theorem 5-4:

Let x and y be vertices in P and P', $depth(x)=depth(y)$. There is an isomorphism from $H[Below(x)]$ to $H'[Below(y)]$ respecting P and P' iff

1. x and y have the same number of children, say k.
2. If $k > 0$, there is a bijection π from $x_1, \ldots, x_k$, the children of x, to $y_1, \ldots, y_k$, the children of y, and for each $i=1, \ldots, k$, there is an isomorphism (θ_i,ϕ_i) from $H[Below(x_i)]$ to $H'[Below(\pi(x_i))]$ that respects P and P'.
3. There is an isomorphism (θ_0,ϕ_0) from $H[x]$ to $H[y]$ that respects P and P' such that, for every $i=1, \ldots, k$, an edge e in $E(H)$ intersects both x and x_i iff its image $g(f)_0(e)$ intersects both y and $\pi(x_i)$.

Proof:

($\Rightarrow$) Let (θ,ϕ) be an isomorphism from $H[Below(x)]$ to $H'[Below(y)]$ that respects P and P'. Let

$\phi_0(i)=\phi(i)$, e_i an edge in $E(H)$ intersecting x,
$\phi_f(i)=\phi(i)$, e_i an edge in $E(H)$ intersecting $Below(x_i)$,

$\theta_0(a)=\theta(a)$, a a vertex in x,
$\theta_f(a)=\theta(a)$, a a vertex in $Below(x_i)$.

Since (θ,ϕ) respects P and P', the map

$\pi(z)=\{\theta(a):a \in z\}$, z a descendant of x in P

is an isomorphism from the subtree of P rooted at x to the subtree of

P' rooted at y, and π restricted to the children of x is a bijection from those children to the children of y.

(θ_0,ϕ_0) is an isomorphism from $H[x]$ to $H'[y]$: since θ maps x to y, θ_0 is a bijection, and since every edge in $E(H)$ intersecting x is mapped to an edge in $E(H')$ intersecting y (and vice versa), ϕ_0 is a bijection from the family $E(H[x])$ to $E(H'[y])$. For every a in x, and every edge e in $E(H)$, $a \in e$ iff $\theta_0(a)=\theta(a) \in \phi(e)$, and since $\phi_0(e)=\phi(e) \cap y$ and $\theta(a) \in y$, $a \in x$ iff $\theta_0(a) \in \phi_0(e)$. Since (θ,ϕ) respects P and P', $deg_H(e)=deg_{H'}(\phi_0(e))$ for any edge e in $E(H[x])$.

(θ_i,ϕ_i) $(i=1, \ldots ,k)$ is a bijection from $H[Below(x_i)]$ to $H'[Below(\pi(x_i))]$. Since π is an isomorphism from the subtree of P rooted at x to the subtree of P' rooted at y, $Below(\pi(x_i)) = \{ a \in \pi(z)) : z$ a descendant of $x_i\}$. So, θ_i is a bijection from $Below(x_i)$ to $Below(\pi(x_i))$. By an argument similar to that for (θ_0,ϕ_0) one can show that ϕ_i is a bijection from $E(H[Below(x_i)])$ to $E(H'[Below(\pi(x_i))])$, and (θ_i,ϕ_i) is in fact an isomorphism respecting P and P'.

Finally, observe that, because (θ,ϕ) is an isomorphism, an edge e in $E(H)$ intersects x and x_i iff the edge $\phi(e)$ intersects y and $Below(\pi(x_i))$.

($\Leftarrow$) Let π, (θ_i,ϕ_0) $(i=0, \ldots ,k)$ be as stated. We build (θ,ϕ) as follows.

$\theta(a)=\theta_0(a)$, if a is a vertex in x,
$\theta(a)=\theta_i(a)$, if a is a vertex in $Below(x_i)$,

$\phi(e)=\phi_i(e)$, if e is an edge in $E(H)$ intersecting $Below(x_i)$,
$\phi(e)=\phi_0(e)$, if e is an edge in $E(H[x])$ not intersecting any x_i.

This definition is consistent because x and the sets $Below(x_i)$ are all disjoint, and because any edge e intersects at most one $Below(x_i)$ (from definition 3-4, part 3). Clearly, θ is a bijection. The function ϕ is a bijection because ϕ_0 restricted to the edges in $E[H]$ that intersect x but not any child of x is a bijection.

(θ,ϕ) is an isomorphism: one need only show that if v is in an edge e (v in $Below(x)$, e in $E(H)$) then $\theta(v)$ is an element of $\phi(e)$. The other case is symmetrical (replace each bijection by its inverse and interchange H and P with H' and P').

1. If v is in x and e does not intersect any $Below(x_i)$ then $\theta(v)=\theta_0(v)$ is an element of $\phi_0(e)=\phi(e)$.
2. If v is in x and e intersects some x_i then $\theta(v)=\theta_0(v)$, $\phi(e)=\phi_i(e)$. We know that
   a. $\phi_0(e)$ and $\phi_i(e)$ intersect y and $Below(\pi(x_i))$,
   b. $|e \cap x|=|\phi_0(e) \cap y|$,
   c. $deg_{H'}(\phi_0(e))=deg_{H'}(\phi_i(e))$, and
   d. $|e \cap Below(x_i)|=|\phi_i(e) \cap Below(\pi(x_i))|$.

   Therefore, $|\phi_i(e) \cap y| = |\phi_0(e) \cap y|$, and since P' is a PPT these

two sets must be equal. Therefore, $\theta(v)=\theta_0(v)$ is an element of $\phi_i(e)=\phi(e)$.

3. If v is in $Below(x_i)$ for some i, then $\theta(v)=\theta_i(v) \in \phi_i(e)=\phi(e)$.

(θ,ϕ) respects P and P': clearly, $deg_H(e)=deg_{H'}(\phi(e))$ for all edges e in $E(H[Below(x)])$, since each (θ_i,ϕ_i) respects P and P'. For the same reason, θ maps vertices u and v in z (z a vertex in $V(P)$) to vertices in the same vertex in P'. #

This theorem will be used to prove the correctness of the directed path graph isomorphism algorithm *DPHIso*, presented below (figures 5-2, 5-3 and 5-4).

Theorem 5-5:

Let Z and Z' be PQR trees, A and A' families of subsets of $FRONTIER(Z)$ and $FRONTIER(Z')$, and lbl and lbl' functions mapping A and A' to labels. $PQRCanon(Z,A,lbl)=PQRCanon(Z',A',lbl')$ iff there is an edge labelled isomorphism from (Z,A) to (Z',A').

Comment Algorithm for determining if there is an isomorphism from H to H' respecting the partial path trees P and P';

proc *DPHIso*(P,P')

begin

1. **if** P and P' do not have the same number of vertices at any depth
2. **then return**(false);
3. d := *depth*(P); /* = *depth*(P') */
4. **for** i := d **downto** 0 **do**
5. D := $\{v|v$ a vertex in P or P', *depth*$[v]=i\}$;
6. **for** each v in A **do** *PPTLABEL*$[v]$:= *Canon*(v);
7. **for** each v in A **do**
8. *PPTINDEX*$[v]$:= the index of *PPTLABEL*$[v]$ in the set
9. $\{$*PPTLABEL*$[u]:u \in A\}$;
10. **return**(*PPTINDEX*[*root*(P)]=*PPTINDEX*[*root*(P')])

end *DPHIso*;

Figure 5-2: Procedure *DPHIso*

Comment Produce a canonical form for $H[x]$ (the case for $H'[x]$ is similar, replace H^x and Z^x with H'^y and Z'^y);

proc *Canon*(x)

begin

11. H^x := $H[x]$+{$e(y,2)$:y a child of x in P, $|CONN(y)|>1$};

12. **for** each edge e in $E(H^x)$ **do**

13. **if** e intersects a child y of x

14. **then** $lbl(e)$:= "(" || $deg_H(e)$ || "," || $PPTINDEX[y]$ || ")"

15. **else** $lbl(e)$:= $deg_H(e)$;

16. Z^x := a PQ tree for H^x;

17. **for** each child y of x, $|CONN(y)|>1$ **do**

18. Z^x := $ORIENT(e(y,2),c(y,1),Z^x)$;

19. **return** ($PQRCanon(Z^x,E(H^x),lbl)$)

end *Canon*;

Figure 5-3: Procedure *Canon*

Comment *PQRCanon* produces a canonical form for edge-labelled PQR trees;

proc *PQRCanon(Z,A,lbl)*

begin

20. Z := *PQRLabel*(Z,A,*lbl*);

21. **return** (*Trav*(Z));

end *PQRCanon*;

proc *Trav*(Z)

begin

22. Let *root*(Z) have children $x_1, \ldots, x_k$;

23. **return** (L[*root*(Z)] || "(" || *Trav*(x_1) || ... || *Trav*(x_k) || ")")

end *Trav*;

Figure 5-4: Procedures *PQRCanon* and *Trav*

Proof:

Examining the proof of theorem 5-3, we see that after running *PQRLabel*, for each node x in Z the children of x are oriented in a way that is independent of their original orientation and that is completely determined (up to isomorphism of the subtrees rooted at the children) by the label L[x]. We can reconstruct the tree Z from the canonical form produced by *PQRCanon*, so if two trees Z and Z' are edge labelled isomorphic they will have the same canonical forms. #

Theorem 5-6:

Let P and P' be PPTs for directed path hypergraphs H and H'. Let H and H' have directed path trees with roots in $root(P)$ and $root(P')$. There is an isomorphism from H to H' respecting P and P' iff $DPHIso(P,P')$ returns true.

Proof:

If P and P' do not have the same number of vertices at any depth then there cannot be an isomorphism from H to H' respecting P and P'.

Let x and y be vertices in P and P', $depth(x)=depth(y)$. We prove by induction on $depth(P)$-$depth(x)$ that there is an isomorphism from $H[Below(x)]$ to $H'[Below(y)]$ respecting P and P' iff $DPHIso$ makes $PPTLABEL[x]=PPTLABEL[y]$.

Let x and y be as above. Assume the induction hypothesis has been proved for all deeper vertices.

($\Leftarrow$) Assume $PPTLABEL[x]=PPTLABEL[y]$. Then, by theorem 5-3, there is an edge labelled isomorphism from the PQR tree for H^x to the PQR tree for H'^y. Because of the choice of edge labels in procedure Canon, this isomorphism maps respects P and P' and maps edges that intersect in $E[H]$ x and some child of u of x to edges that intersect in $E[H']$ y and some child v of y. Moreover, if an edge of this second type

intersects u and is mapped to an edge intersecting v then $PPTINDEX[u]$ $=PPTINDEX[v]$, so $PPTLABEL[u]=PPTLABEL[v]$ and, by the induction hypothesis, there is an isomorphism from $H[Below(u)]$ to $H'[Below(v)]$ respecting P and P'.

We are unable to immediately apply theorem 5-4 because the isomorphism from H^x to H^y may map edges that are in the same connector in P to edges that are in different connectors in P'. We show that in this case the connectors are identical.

Let u be a child of x and v and w children of y, and let $PPTLABEL[u]$ $=PPTLABEL[v]=PPTLABEL[w]$. The connectors of u, v and w are therefore isomorphic. Let (θ,ϕ) be an isomorphism from H^x to H'^y preserving edge labels. Let e and f be edges that intersect both x and u in H, and suppose that $\phi(e)$ intersects y and v in H' while $\phi(f)$ intersects y and w.

If $CONN(u)$ is trivial then so are $CONN(v)$ and $CONN(w)$, and since $e=f$, $\phi(e)=\phi(f)$, so $CONN(v)$ and $CONN(w)$ are identical.

If $CONN(u)$ is not trivial then the characteristic nodes of $e(u,1)$, $e(v,1)$ and $e(w,1)$ in their PQR trees must be R nodes. Since the $PPTLABEL$'s of u, v and w are identical, $|CONN(u)|=|CONN(v)|=|CONN(w)|=k$ >1. Let $e \cap x=c(u,i)$, $f \cap x=c(u,j)$ (assume $j \leq i$, otherwise, swap e and f and v and w). Since each connector is oriented outwards, the

vertices in $c(u,i)$ are the final $|c(u,i)|$ vertices in $e(u,1)$ in any σ in $CONSISTENT(Z^x)$ (similarly for v and w). We conclude that the final $|c(u,j)|$ vertices in $e(v,1)$ and $e(w,1)$ are the same in any σ in $CONSISTENT(Z'^y)$, and therefore $e(v,1)=e(w,1)$, so the connectors are identical.

Therefore, two edges e and f in the same connector $CONN(u)$ are mapped to different connector $CONN(v)$ and $CONN(w)$ iff $CONN(v)$ and $CONN(w)$ are identical. By suitably modifying ϕ we can ensure that e and f intersect some child u of x iff $\phi(e)$ and $\phi(f)$ intersect some child v of y, $PPTLABEL[u]=PPTLABEL[v]$. We can now use theorem 5-4 to conclude that there is an isomorphism from $H[Below(x)]$ to $H'[Below(y)]$ respecting H and H'.

($\Rightarrow$) Assume there is an isomorphism from $H[Below(x)]$ to $H'[Below(y)]$ respecting P and P'. By theorem 5-4, x and y have the same number of children, say k, there is a bijection π from the children of x, $x_1, \ldots, x_k$ to the children of y and a collection of isomorphisms (θ_i,ϕ_i) $(i=1, \ldots, k)$ respecting P and P' from $H[Below(x_i)]$ to $H'[Below(\pi(x_i))]$, and there is an isomorphism (θ_0,ϕ_0) from $H[x]$ to $H[y]$ respecting P and P' such that for every edge e in $E[H]$ intersecting x and x_i, $\phi(e)$ intersects y and $\pi(x_i)$.

Since H and H' have directed path trees with roots in $root(P)$ and $root(P')$, they have path trees in which all the connectors $CONN(x_i)$

($CONN(\pi(x_i))$ are oriented outwards. Therefore, there exist PQR trees Z^x and Z'^y for H^x and H'^y in which all connectors are oriented outwards; the function θ_0 from x to y can be extended to an isomorphism from Z^x to Z'^y. Moreover, because of the three conditions of theorem 5-4, and because $PPTINDEX[x_i]$=$PPTINDEX[\pi(x_i)]$, clearly this also yields an edge labelled isomorphism from Z^x to Z'^y (and the edge labels as given in *Canon*). #

6. Conclusions and Directions for Future Research

Linear time algorithms for finding partial path trees and directed path trees for directed path hypergraphs have been presented. Together with a linear time algorithm for extracting the maximal cliques of a chordal graph, this yields a linear time algorithm for directed path graph recognition.

A polynomial time algorithm for directed path graph isomorphism was presented. The algorithm used canonical labelling ideas from tree isomorphism, and can be thought of as a combination of the usual tree isomorphism algorithm and Lueker and Booth's interval graph isomorphism algorithm. This may not be surprising, as the class of directed path graphs include both trees and interval graphs.

The algorithms *Find-PPT* and the procedure for finding directed path trees from PPT's are overly complicated; it would be nice to have simpler algorithms.

An open problem is to find a linear time isomorphism algorithm for directed path graphs. There is no obvious way to modify *DPHIso* to run in linear time, because (unlike in trees, where the center may be used) there is no obvious isomorphism-invariant way to find a set of vertices in a directed path hypergraph that form a root of a PPT. A possible direction of research is to modify *Find-PPT* so that it discards vertices in the same way on isomorphic hypergraphs.

Call a hypergraph $H=(V,E)$ *out-degree one* if there is a directed graph $D=(V,A)$

in which all vertices have out-degree 1 and in which every edge in E induces a directed path. Truszczynski has given a polynomial time algorithm for recognizing these hypergraphs [42]; perhaps the techniques given in this thesis can be extended to yield a linear time algorithm, just as PQ trees can used to solve the problem of recognizing circular arc hypergraphs.

Another open problem is to find a linear time algorithm for recognizing undirected path graphs. It is known that isomorphism of undirected path graphs is isomorphism-complete.

Booth and Johnson have given a linear time algorithm that, given a characteristic tree for a directed path graph, finds a maximum dominating set for the graph [8]. It would be interesting to find other problems that are NP-complete on chordal graphs yet can be solved efficiently on directed path graphs.

The complexity of the hamiltonian circuit problem is unknown on chordal graphs and interval graphs. Fast algorithms or proofs of NP-completeness would be interesting.

References

[1] Aho, A.V., Hopcroft, J.E. and Ullman, J.D.
The Design and Analysis of Computer Algorithms.
Addison-Wesley, Reading, MA, 1974.

[2] Benzer, S.
On the Topology of the Genetic Fine Structure.
Proceedings of the National Academy of Sciences 45:1607-1620, 1959.

[3] Bertele, U. and Brioschi, F.
Mathematics in Science and Engineering. Volume 91: *Nonserial Dynamic Programming.*
Academic Press, New York, 1972.

[4] Berge, C.
Graphs and Hypergraphs.
North Holland, New York, 1973.

[5] Bertossi, A.A.
Finding Hamiltonian circuits in proper interval graphs.
Information Processing Letters 17:97-101, 1983.

[6] Bondy, J.A. and Murty, U.S.R.
Graph Theory with Applications.
North Holland, New York, 1976.

[7] Booth,K.S. and Colbourn,C.J.
Problems polynomially equivalent to graph isomorphism.
Technical Report CS-77-04, University of Waterloo, 1979.

[8] Booth, K.S. and Johnson, J.H.
Dominating Sets in Chordal Graphs.
SIAM Journal of Computing 11(1):191-199, February, 1982.

[9] Booth, K.S. and Lueker, G.S.
Testing For the Consecutive Ones Property, Interval Graphs and Graph Planarity Using PQ-Tree Algorithms.
Journal of Computer and System Sciences 13:335-379, 1976.

[10] Booth, K.S.
PQ-Tree Algorithms.
PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1975.

[11] Buneman, P.
A Characterization of Rigid Circuit Graphs.
Discrete Mathematics 9:205-212, 1974.

[12] Busacker, R.G. and Saaty, T.L.
Finite Graphs and Networks.
McGraw-Hill, New York, 1965.

[13] Colbourn, C.J. and Booth, K.S.
Linear Time Automorphism Algorithms for Trees, Interval Graphs, and Planar Graphs.
Technical Report CS-79-06, University of Waterloo, 1980.

[14] Diamond, J.S. and Booth, K.S.
Implementation of a Linear Time Isomorphism Test for Interval Graphs.
In Yahiko Kambayashi (editor), *Proceedings of the Conference on Consecutive Retrieval Property: Theory and Applications*, pages 56-67.
ICS Polish Academy of Sciences, Warsaw, Poland, July, 1981.

[15] Dietz, P.F., Furst, M. and Hopcroft, J.E.
A Linear Time Algorithm for the Directed Tree Problem and Directed Path Graphs.
1984.

[16] Even, S.
Graph Algorithms.
Computer Science Press, Rockville, MD, 1979.

[17] Fujishige, S.
An Efficient PQ-Graph Algorithm for Solving the Graph-Realization Problem.
Journal of Computer and System Sciences 21(1):63-86, 1980.

[18] Fulkerson, D.R. and Gross, O.A.
Incidence Matrices and Interval Graphs.
Pacific Journal of Mathematics 15:835-855, 1965.

[19] Garey, M.R. and Johnson, D.S.
Computers and Intractability: A Guide to the Theory of NP-Completeness.
Freeman, San Francisco, 1979.

[20] Gavril, F.
Algorithms for Minimum Coloring, Maximum Clique, Minimum Covering by Cliques, and Maximum Independent Set of a Chordal Graph.
SIAM Journal of Computing 1:180-187, 1972.

[21] Gavril, F.
The Intersection Graphs of Subtrees in Trees Are Exactly the Chordal Graphs.
J. of Combinatorial Theory (B) 16:47-56, 1974.

[22] Gavril, F.
A Recognition Algorithm for the Intersection Graphs of Directed Paths in Directed Trees.
Discrete Mathematics 13:237-249, 1975.

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

[24] Gavril, F. and Tamari, R.
An Algorithm for Constructing Edge-Trees from Hypergraphs.
Networks 13:377-389, 1983.

[25] Ghosh, S.P.
File Organization: The Consecutive Retrieval Property.
Communications of the ACM 15:802-808, 1972.

[26] Gilmore, P.C. and Hoffman, A.J.
A Characterization of Comparability Graphs and of Interval Graphs.
Canadian Journal of Mathematics 16:539-548, 1964.

[27] Gilbert, J.R. and Rose, D.J.
A Separator Theorem for Chordal Graphs.
Technical Report TR 82-523, Cornell University Department of Computer Science, October, 1982.

[28] Golumbic, M.C.
Algorithmic Graph Theory and Perfect Graphs.
Academic Press, New York, 1980.

[29] Hoffmann, C. M.
Lecture Notes in Computer Science. Volume 136: *Group-Theoretic Algorithms and Graph Isomorphism.*
Springer-Verlag, New York, 1982.

[30] Kashiwabara, T.
Algorithms for Some Intersection Graphs.
In Saito, N. and Nishizeki, T. (editors), *Graph Theory and Algorithms.* Springer-Verlag, October, 1980.

[31] Lekkerkerker, C.G. and Boland, J. Ch.
Representation of a Finite Graph By a Set of Intervals on the Real Line.
Fundamenta Mathematicae 51:45-64, 1962.

[32] Lipski, W. Jr.
The Consecutive Retrieval Property, Interval Graphs and Related Topics - a Survey.
In Yahiko Kambayashi (editor), *Proceedings of the Conference on Consecutive Retrieval Property: Theory and Applications*, pages 110-141. ICS Polish Academy of Sciences, Warsaw, Poland, July, 1981.
Contains an extensive bibliography on problems related to the consecutive retrieval problem.

[33] Lipton, R.J. and Tarjan, R.E.
Applications of a Planar Separator Theorem.
In *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*, pages 162-170. IEEE Computer Society, 1977.

[34] Lipton, R.J. and Tarjan, R.E.
A Separator Theorem for Planar Graphs.
SIAM J. of Appl. Math. 36(2):177-189, April, 1979.

[35] Lueker, G.S. and Booth, K.S.
A Linear Time Algorithm for Deciding Interval Graph Isomorphism.
Journal of the ACM 26(2):183-195, April, 1979.

[36] Lueker, G.S.
Efficient Algorithms for Chordal Graphs and Interval Graphs.
PhD thesis, Program in Applied Mathematics and the Department of Electrical Engineering, Princeton University, 1975.

[37] Luks, E.M.
Isomorphism of Graphs of Bounded Valence Can be Tested in Polynomial Time.
In *Proceedings of the 21st Annual Symposium on the Foundations of Computer Science*, pages 42-49. IEEE Computer Society, October, 1980.

[38] Mirkin, B.G. and Rodin, S.N.
Biomathematics. Volume 11: *Graphs and Genes.*
Springer-Verlag, New York, 1984.

[39] Rose, Donald J.
Triangulated Graphs and the Elimination Process.
Journal of Mathematical Analysis and Applications 32:597-609, 1970.

[40] Rose, D.J., Tarjan, R.E. and Lueker, G.S.
Algorithmic Aspects of Vertex Elimination on Graphs.
SIAM Journal of Computing 5:266-283, 1976.

[41] Tanaka, K.
Tree-Structured Data Organization With Consecutive Retrieval Property.
In Yahiko Kambayashi (editor), *Proceedings of the Conference on Consecutive Retrieval Property: Theory and Applications*, pages 220-225. ICS Polish Academy of Sciences, Warsaw, Poland, July, 1981.

[42] Truszczynski, M.
On Admissible Families of Sets.
In Yahiko Kambayashi (editor), *Proceedings of the Conference on Consecutive Retrieval Property: Theory and Applications*, pages 246-266. ICS Polish Academy of Sciences, Warsaw, Poland, July, 1981.

[43] Truszczynski, M.
On Acyclic Consecutive Retrieval Organizations.
In Yahiko Kambayashi (editor), *Proceedings of the Conference on Consecutive Retrieval Property: Theory and Applications*, pages 267-276. ICS Polish Academy of Sciences, Warsaw, Poland, July, 1981.

[44] Truszczynski, M.
The Theorem Characterizing the Acyclic Families of Sets.
Technical Report 314, ICS Polish Academy of Sciences, 1978.

[45] Young, S.M.
Implementation of PQ-Tree Algorithms.
Master's thesis, Department of Computer Science, U. of Washington, 1977.

Index

Visit 43, 44

Visited 43

Walk 5