HYPERGRAPH COVERING ALGORITHMS
FOR RELATIONAL QUERY PROCESSING

by

A. Lustig and O. Shmueli

Technical Report #368
May 1985
ABSTRACT

Tree schemas (acyclic hypergraphs) play a fundamental role in the theory of relational query processing. Consider a query \( Q \) which is the join of some relations projected on a prescribed set of attributes. Suppose \( Q \) is solved by a finite program \( P \) using the operators project, join and semijoin. It is known that \( P \) must transform the original set of relations into a new set of relations that contains an "embedded" tree schema which includes the original relations and the desired result relation; such a schema is called a tree projection.

This work analyzes two classes of simple cyclic schemas called Arings and Diamonds. Given database schema \( D_1 \) (which is an Aring or a Diamond) and an arbitrary database schema \( D_2 \), polynomial time algorithms are presented for deciding whether there exists a tree projection. A tree projection is a tree schema \( D_3 \) such that each relation schema in \( D_3 \) is contained in some relation of \( D_2 \) and each relation in \( D_1 \) is contained in some relation of \( D_3 \). These algorithms are not merely of theoretical interest; they may prove useful as part of query processors for both centralized and distributed database systems.
1. INTRODUCTION

Since its introduction by Bernstein, Chiu and Goodman [BC, BG], acyclicity has emerged as a basic tool in relational database theory. The partitioning of schemas into tree (acyclic) and cyclic schemas has many important consequences for dependency theory and database design theory [BFMMUY, BFMY, Hul, Kat, LMG, SMF]. Classification of queries into those implying a cyclic schema versus those implying a tree schema was one of the research avenues pursued [Fag, GS1-GS5, GST, KY, KYY, Yan, YO].

Tree schemas play a fundamental role in relational query processing. Consider a query $Q$ which is the join of some relations projected on a prescribed set of attributes. Suppose $Q$ is solved by a finite program $P$ using the operators project, join and semijoin (the semijoin of relation $R$ into $S$ is the join of $R$ and $S$ projected onto $S$'s attributes). It is shown in [GS3] that $P$ must transform the original set of relations into a new set of relations that contains an "embedded" tree schema which includes the original relations and the desired result relation; such a schema is called a tree projection (precise definitions are given in Section 2).

The problem of deciding whether a tree projection exists is still open (in the general case). This is basically a hypergraph (schema) theoretic problem: given two hypergraphs (schemas) $H_1$ and $H_2$ is there an acyclic hypergraph (tree schema) $H_3$ such that each edge (relation schema) in $H_3$ is contained in some edge of $H_2$ and each edge in $H_1$ is contained in some edge of $H_3$ (see [Ber] regarding Hypergraphs). The above problem is clearly in $NP$ (see [GJ] for definitions). Polynomial time algorithms are presented for solving this problem when $H_1$ above is either an Aring or a Diamond.

These algorithms are not merely of theoretical interest. They may be used in query processors in both centralized and distributed database systems. Identifying the existence of tree projections provides new heuristics for processing queries. A detailed discussion of this topic appears in Appendix A.
This paper is organized as follows. Section 2 presents basic concepts and definitions. In Section 3 and 4, polynomial time algorithms are presented for identifying a tree projection with respect to Aring and Diamond schemas, respectively. Section 5 presents conclusions and new problems.
2. TERMINOLOGY

2.1. Relational Databases

A universe \( U \) is a finite set of attributes. A relation schema (or simply relation) \( R_i \) is a subset of \( U \), and a database schema \( D \) (or simply schema) is a multiset of relation schemas\(^1\). Clearly, a database schema may be viewed as the set of edges of a hypergraph over \( U \) [Ber]. Associated with each \( A \in U \) is a possibly infinite domain, \( \text{dom}(A) \). The domain of a relation schema \( R_i = [A_1, \ldots, A_{n_i}] \) is \( \text{dom}(R_i) = \prod_{k=1}^{n_i} \text{dom}(A_k) \). The length of \( R_i \), denoted \( |R_i| \), is \( n_i \). The length of schema \( D \) is \( |D| = \sum_{R_i \in D} |R_i| \). Let \( \cup(D) \) denote the set of attributes in schema \( D \).

A relation state \( R_i \) for relation schema \( R_i \) is a finite subset of \( \text{dom}(R_i) \); one can think about the state as a table of data with columns \( A_1, \ldots, A_{n_i} \). A database state for schema \( D \) is an assignment of relation states to \( D \)'s relation schemas. We use \( D = (R_1, \ldots, R_n) \) to denote a database schema and \( D = (R_1, \ldots, R_n) \) for a corresponding state. Elements in a relation state are called tuples. Tuple \( t \) over schema \( R \) matches tuple \( s \) over schema \( S \) if for all \( A \in R \cap S \), the values of tuples \( t \) and \( s \) for attribute \( A \) are identical.

The projection of relation state \( R \) over attribute set \( X \subseteq R \), denoted \( R[X] \) or \( \pi_X(R) \), is the maximal subset of \( \text{dom}(X) \) containing tuples that match some tuple in \( R \). The (natural) join of relation states \( R \) and \( S \), denoted \( R \Join S \), is defined as the maximal subset in \( \text{dom}(R \cup S) \) containing tuples that match a tuple in \( R \) and a tuple in \( S \). The (natural) semi-join of relation states \( R \) and \( S \), denoted \( R \Join S \) is defined as \( \pi_X(R \Join S) \). For a database \( D \) over schema \( D \) define \( J(D) = \bigcup_{R_i \in D} R_i \).

2.2. Tree Schemas

Let \( G = (V,E) \) be an undirected graph\(^2\) whose nodes are in one-to-one

---
\(^1\) All structures in this paper are finite.
\(^2\) We use traditional graph theory notation.
correspondence with the relations of schema $D$. We say $G$ is $A$-connected if the subgraph of $G$ induced by relations (nodes) containing attribute $A$ is connected. $G$ is $X$-connected if for all $A \in X$, $G$ is $A$-connected. $G$ is a qual graph for $D$ if it is $\cup(D)$-connected [BG]. This property of a qual graph is called attribute connectivity. A qual graph for scheme $D$ is minimal if there exists no other qual graph for $D$ with a smaller number of edges. Two relations of schema $D$ are overlapping if they adjacent in every minimal qual graph for $D$.

$D$ is a tree schema (acyclic hypergraph) if some qual graph for it is a tree; otherwise $D$ is a cyclic schema (cyclic hypergraph). An attribute $A \in \cup(D)$ is isolated if it appears in exactly one relation schema. The following simple procedure, called Graham reduction and discovered independently by [Gra] and [YO], recognizes tree schemas.

Procedure $GR$

Apply the following two steps until neither is applicable:

1. **Step 1**: Delete any isolated attribute.
2. **Step 2**: Find two relation schemas $R$ and $S$ in $D$ such that $R \subseteq S$; delete $R$ from $D$.

We denote by $GR(D)$ the output of procedure $GR$ on input of schema $D$. It can be shown that $D$ is a tree schema iff upon termination of the above procedure the resulting schema consists of a single (empty) relation schema, i.e. $GR(D) = (\emptyset)$. (A linear time algorithm for recognizing tree schemas appears in [TY].)

Example 2.1

The following is a tree schema: $D_1 = \{(1,2),(1,3),(1,4),(1,5),(2,3),(2,4),(2,5),(3,4),(3,5),(4,5)\}$. A qual graph for $D_1$ is:

```
{1,2} -- {1,2,3} -- {1,4,5} -- {1,4}
   |           |            |
{2,3}    {1,5}
```

The schema $D_2 = \{(1,2),(2,3),(3,4),(4,1)\}$ is cyclic. Any qual graph for $D_2$ is iso-

---

$^3$ When no confusion may arise we let $V = D$ and talk about "node $R_i"$ rather than "node $v_i \in V$ corresponding to $R_i \in D"$.
2.3. Tree Projections

\( D \) is a projection of \( D' \), denoted \( D \leq D' \), if for every relation \( R \in D \) there exists a relation \( R' \in D \) such that \( R \subseteq R' \). Let \( D \) and \( D' \) be schemas such that \( D \leq D' \). We say that \( D' \) is a tree projection (TP) of \( D' \) with respect to (w.r.t.) \( D \), written \( D' \in TP(D',D) \), if \( D' \) is a tree schema and \( D \leq D' \leq D' \) [GS3]. When \( D' \in TP(D',D) \), i.e. \( D' \) is a TP of itself w.r.t. \( D \), we usually just say \( D' \) is a TP w.r.t. \( D \). If \( D' \in TP(D',D) \) then, because \( D \leq D' \), without loss of generality (w.l.o.g.) we may assume that \( D \subseteq D' \). In the remainder of this paper we use \( D = (R_1, \ldots, R_n) \) and \( D' = (R_1, \ldots, R_n, R_{n+1}, \ldots, R_m) \).

We call \( R_i \) (\( 1 \leq i \leq n \)) base relations; \( R_i \) (\( n+1 \leq i \leq m \)) are called non-base relations.

Example 2.2

Let \( D = (\{1,2\}, \{2,3\}, \{3,4\}, \{4,5\}, \{5,6\}, \{6,7\}, \{7,8\}, \{8,1\}) \)

\( D' = D \cup (\{1,2,3,8\}, \{3,4,7,8\}, \{4,5,6,7\}) \)

\( D' = (\{1,2,5,6\}, \{1,2,3,5,8\}, \{1,3,4,7,8\}, \{4,5,6,7\}, \{5\}) \)

Clearly both \( D \) and \( D' \) are cyclic, and \( D' \) is a tree schema. \( D \leq D' \leq D' \), and therefore \( D' \) is a TP of \( D' \) w.r.t. \( D' \).

Let \( X \subseteq \cup(D) \). The following operations map a schema \( D \) into a projection of \( D \).

Uniform deletion of \( X \) from \( D \) is \( del(D,X) = (R_1 \setminus X, R_2 \setminus X, \ldots, R_n \setminus X) \).

Uniform restriction of \( X \) in \( D \) is \( rst(D,X) = (R_1 \cap X, R_2 \cap X, \ldots, R_n \cap X) \).

If \( D \) is a tree schema then \( del(D,X) \) and \( rst(D,X) \) are also tree schemas.

Consider a tree projection \( D' \) w.r.t. schema \( D \). \( D' \) is a minimal tree projection if it satisfies all the following properties:

1. Each base relation appears exactly once in \( D' \).
2. No non-base relation \( R \) is contained in another relation \( S \subseteq D' \).
3. All attributes in non-base relations in \( D' \) are essential, i.e. if any attribute is removed from a non-base relation then the resulting schema is cyclic.
Given a tree projection \( D' \) w.r.t. schema \( D \), it is always possible to obtain a minimal tree projection \( D'' \) w.r.t. \( D \) by repeatedly removing base relations which appear more than once, non-base relations which are subsets of other relations and non-essential attributes in non-base relations. Clearly, \( D'' \subseteq D' \).

**Example 2.3**

Let \( D = \{1,2\}, \{2,3\}, \{3,4\}, \{4,5\}, \{5,6\}, \{6,1\} \) and let schema \( D'' = D \cup \{1,3\}, \{1,2,3,5\}, \{1,4,5,6\}, \{1,3,4,5,6\} \) be a tree projection w.r.t. \( D \). Clearly \( D'' \) is not minimal. Relation \{1,3\}, which is not a base relation, is a subset of relation \{1,2,3\}. Attribute 5 in relation \{1,2,3,5\} is non-essential. A minimal tree projection w.r.t. \( D \) obtained from \( D'' \) is \( D'' = D \cup \{1,3\}, \{1,2,3,5\}, \{1,3,4,5,6\} \). No non-base relation is a subset of another relation and deleting any attribute of a non-base relation results in a cyclic schema.

### 2.4. Programs

A program \( P \) over a schema \( D = (R_1, R_2, \ldots, R_n) \) is a sequence of statements. Each statement creates a new relation. Once a relation is created it is never altered. The \( k \)’th statement is one of the following:

1. **Project statement**: \( R_{n+k} := \pi_X(R_i), \ 1 \leq i \leq n+k-1 \)
2. **Join statement**: \( R_{n+k} := R_i \bowtie R_j, \ 1 \leq i,j \leq n+k-1 \)
3. **Semi-join statement**: \( R_{n+k} := R_i \bowtie R_j, \ 1 \leq i,j \leq n+k-1 \)

A program \( P \) composed of \( p \) statements over schema \( D \) constructs the schema \( D' = (R_1, \ldots, R_n, \ldots, R_{n+p}); D' \) is obtained from \( D \) by \( P \).

Programs may be used for solving queries (in fact they may be seen as an implementation language for relational algebra). The class of queries we consider are the simple natural join queries (SNJ). An SNJ query has the form \( \pi_X(J(D)) \), where \( X \) is contained in some \( R \in D \). SNJ appears to be rather limited; however, a large subclass of the equijoin queries may be efficiently mapped into SNJ [BG].
3. QUERIES OVER ARINGS

3.1. Simple Cyclic Schemas

An *Aring* (of size $n$) is a schema isomorphic to the schema $(\{A_1, A_2\}, \{A_2, A_3\}, \ldots, \{A_{n-1}, A_1\})$ for distinct attributes $A_1, \ldots, A_n$ and $n \geq 3$. Clearly, Arings are cyclic schemas \footnote{Arings, together with another type of cyclic schemas called *Acliques*, are basic “building blocks” for all cyclic schemas [GS2].}. It has been shown that if a program $P$ solves a SNJ query over an Aring schema $D$, then $P$ must produce a schema $D'$ such that there exists $D' \in TP(D', D)$ [GS3]. This holds even if we allow "programs" with an unlimited number of semijoin statements [Kla].

3.2. Properties of Tree Projections over Arings

A *3-tree schema* is a tree schema with no more than three attributes in any relation. Let $3TP$ denote a TP which is also a 3-tree schema and let $m3TP$ denote a minimal 3TP. A schema $D$ is 2-attribute connected if in all qual graphs for $D$, the intersection of every two adjacent relations has cardinality 2.

**Lemma 3.1**

Let $D$ be the Aring over attributes $\{1, 2, \ldots, n\}$, $D'$ a TP w.r.t. $D$, and $T$ a qual tree for $D'$. For any two base relations $\{p, p+1\}$ and $\{q, q+1\}$, let $\{q, q+1\} = S_0, S_1, \ldots, S_k, S_{k+1} = \{p, p+1\}$ be the relations on the path from $\{q, q+1\}$ to $\{p, p+1\}$ in $T$. Let $U_1 = \{q+1, \ldots, p\}$ and $U_2 = \{p+1, \ldots, q\}$ ($n+1=1$). Then, for $0 \leq i \leq k+1$, $S_i \cap U_1 \neq \emptyset$ and $S_i \cap U_2 \neq \emptyset$.

**Proof**

The Lemma clearly holds for $i=0$ and $i=k+1$. W.l.o.g. let $\{q, q+1\} = \{n, 1\}$. For $1 \leq i \leq k$, root $T$ at relation $S_{i-1}$, and let $U_3$ be the attributes contained in all those base relations that are descendants of $S_i$ in $T$ ($S_i$ included). $U_3 \neq \emptyset$ since it contains at least $p$ and $p+1$. Let $A_e$ be the smallest attribute in $U_3$. If $A_e = 1$, then $1 \in S_i$ and the first part of the Lemma holds. If not, consider base relation $\{A_{e-1}, A_e\}$. Because $p \in U_3$, $A_e \leq p$. As $A_{e-1} \notin U_3$, $\{A_{e-1}, A_e\}$ is not a descendant of $S_i$. Because $T$ is $A_e$-connected, $A_e \in S_i$, thus proving the first part of the Lemma. Now let $A_e$ be
the largest attribute in \( U_3 \). If \( A_p = n \), then \( n \in S_k \) and the second part of the Lemma holds. If not, consider base relation \( \{A_p, A_{p+1}\} \). Again we get \( A_p \in S_k \). Since \( A_p \geq p+1 \), the second part of the Lemma holds too. 

Lemma 3.2

If \( D' \) is a m3TP w.r.t. an Aring then \( D' \) is 2-attribute connected.

Proof

Let \( T \) be a qual tree for \( D' \), and let \( S_1 \) and \( S_2 \) be any two relations adjacent in \( T \). Since \( D' \) is a 3-tree schema, the intersection of any two adjacent relations is of cardinality \( \leq 3 \). If \( |S_1 \cap S_2| = 3 \), then \( S_1 = S_2 \), contradicting the fact that \( D' \) is minimal. Root \( T \) at relation \( S_2 \). Let \( T_1 \) be the subtree of \( T \) rooted at relation \( S_1 \), and let \( T_2 \) be \( T - T_1 \), i.e. the tree rooted at \( S_2 \) obtained from \( T \) by disconnecting \( S_1 \) and its subtree. By minimality of \( D' \), \( T_1 \) (resp. \( T_2 \)) must include at least one base relation, say \( \{p, p+1\} \) (resp. \( \{q, q+1\} \)). \( |S_1 \cap S_2| \geq 2 \) follows immediately from Lemma 3.1; since \( |S_1 \cap S_2| < 3 \) we conclude that \( |S_1 \cap S_2| = 2 \).

Lemma 3.3

Let \( D' \) be a m3TP w.r.t. an Aring \( D \). In any qual tree \( T \) for \( D' \), rooted at a non-base relation, all internal nodes are relations of size 3 and the leaves are exactly the base relations from \( D \).

Proof

Suppose an internal node in \( T \) is a relation \( R_i = \{A_p, A_{p+1}\} \) of size 2. Because \( R_i \) is not a leaf it has (at least) two adjacent relations, \( R_j \) and \( R_k \). By Lemma 3.2, the intersection of \( R_i \) and its adjacent relations is of cardinality 2. Therefore, \( A_p \in R_j \) and \( A_{p+1} \in R_k \). Suppose \( R_i \) is a base relation. Since in addition to \( R_i \) there is only one other base relation in the Aring which contains attribute \( A_p \), it is clearly not essential in either \( R_j \) or \( R_k \), contradicting the minimality of \( D' \). Now suppose that \( R_i \) is not a base relation: It follows from \( |R_i \cap R_j| = 2 \) that \( R_i \not\subset R_j \), again contradicting the minimality of \( D' \). Because base relations are of size 2, they must be leaves of \( T \). Suppose a leaf of \( T \) is not a base relation; clearly, all the attributes of that leaf relation are not essential, contradicting the minimality of \( D' \).
Lemma 3.4

Let D be an Aring of size n where w.l.o.g. \( R_k = [A_k, A_{k+1}] \) and \( D' \) a TP w.r.t. D. There is some relation \( S \in D' \) and index \( 1 \leq k \leq n \), such that \( R_k \subseteq S \) and \( R_{k+1} \subseteq S \).

Proof

Consider any qual tree \( T \) for \( D' \), and root it at any non-base relation. For \( i=1..n \), \( R_i \) and \( R_{i+1} \) have a lowest common ancestor \( LCA_i \) in \( T \). Observe that \( LCA_i \) must be on the unique path connecting \( R_i \) and the root. Therefore, either \( LCA_i = LCA_{i+1} \), or \( LCA_i \) is an ancestor of \( LCA_{i+1} \), or \( LCA_{i+1} \) is an ancestor of \( LCA_i \).

Construct a directed graph \( G = ((1,2,...,n), E) \), where for \( 1 \leq i \leq n \) the edges \( E \) are:

- \((i,i+1)\) if \( LCA_i = LCA_{i+1} \),
- \((i,i+1)\) if \( LCA_{i+1} \) is a descendant of \( LCA_i \),
- \((i+1,i)\) if \( LCA_i \) is a descendant of \( LCA_{i+1} \).

Examine the possibilities for cycles in \( G \) of the form \((1,2), (2,3), ..., (n-1,n), (n,1)\).
If \( G \) has such a cycle then all \( LCA_i \) are identical to a single relation \( S \) and by LCA definition \( S \) must include all attributes. The Lemma clearly holds for any index \( k \).
Otherwise, since each vertex is adjacent to two directed edges, there exists a vertex \( k \) with two ingoing edges \((k+1,k)\) and \((k-1,k)\). Because \( LCA_k \) is an ancestor of \( R_{k+1} \), and \( LCA_{k+1} \) an ancestor of \( LCA_k \), we have \( A_k \in LCA_k \), and of course \( A_{k+1} \in LCA_k \). Therefore \( \{A_k,A_{k+1}\} \subseteq S \) and \( \{A_{k+1},A_{k+2}\} \subseteq R_{k+1} \subseteq S \).

Example 3.1

Let \( D \) be the Aring over attributes \{1,2,...,8\} and \( D' = D \cup \{1,2,3,\},\{2,3,8\},\{3,4,5\},\{1,2,3,6\},\{3,4,5,6\}\}. Clearly, \( D' \) is a TP w.r.t. D. A possible qual tree for \( D' \), rooted at relation \( \{2,3;8\} \) is shown in Figure 3.1. The graph \( G \) for this tree, as constructed in the proof of Lemma 3.4, is given in Figure 3.1. Relations \( \{3,4,5,8\} \) and \( \{1,2,3,8\} \) have the properties of relation \( S \) in Lemma 3.4.

---

\(^5\) \( n+1 \rightarrow 1 \).

\(^6\) Figures designated En appear at the end of this paper.
Corollary 3.1

Let $D'$, $T$, $R_k$, $R_{k+1}$ and $S$ be as in Lemma 3.4. If $D'$ is minimal then $R_k$ and $R_{k+1}$ are adjacent to $S$ in $T$, and the only relations in $D'$ containing $R_{k+1}$ are $R_k$, $R_{k+1}$ and $S$.

**Proof**

Root $T$ at $S$ and suppose that $R_k$ and $R_{k+1}$ are not children of $S$. Let $S_1$ be the parent of $R_k$ in $T$, observe that $R_{k+1} \in S_1$. If $R_k$ is an ancestor of $R_{k+1}$ in $T$, then by attribute connectivity requirements in $T$, $R_k \supseteq \{A_k, A_{k+1}\}$, which is impossible. Similarly, $R_{k+1}$ cannot be an ancestor of $R_k$. Modify $T$ by attaching $R_k$ and $R_{k+1}$ to $S$ instead of their original parents, call the resulting tree $T'$. Consider descendants of $S_1$ in $T'$. Let $S_2$ be the lowest descendant of $S_1$ (it may also be $S_1$ itself) containing $R_{k+1}$. Clearly, $R_{k+1}$ is non-essential in $S_2$, contradicting the minimality of $D'$.

**Example 3.2**

Let $D$ and $D'$ be as in Example 3.1. A minimal TP obtained from $D'$ is $D'' = D \setminus \{\{1,2,6\}, \{2,3,6\}, \{3,4,5,6\}\}$. A qual tree for $D''$, rooted at relation $\{2,3,6\}$, is given in Figure 3.2. Note that relations $\{3,4\}$ and $\{4,5\}$ are adjacent to relation $\{3,4,5,6\}$, and relations $\{6,1\}$ and $\{1,2\}$ are adjacent to relation $\{1,2,6\}$.

The following two Lemmas shed more light on the structure of tree projections w.r.t. Arrings. These Lemmas are used in the next section and the reader may skip them at this point.

---

Figure 3.1
Lemma 3.5 (see Appendix)
Let $D$ and $D'$ be as in Lemma 3.4. There exist two relations $S_1 \subseteq D'$ and $S_2 \subseteq D'$ (not necessarily distinct), and two indices $k_1$ and $k_2$ such that $R_{k_1} \subseteq S_1$, $R_{k_1+1} \subseteq S_1$, $R_{k_2} \subseteq S_2$, and $R_{k_2+1} \subseteq S_2$. If $p > 3$ then $k_1$ and $k_2$ can be chosen so that $|k_1 - k_2| > 1$. 

Lemma 3.6 (see Appendix)
Let $D$ be an Aring, $D'$ a TP w.r.t. $D$, $T$ a qual tree for $D'$ rooted at some relation and $T_1$ a subtree of $T$. Let $G$ be a sequence of two or more overlapping base relations $\{i,i+1\}, \{i+1,i+2\}, \ldots, \{j-1,j\}$ appearing in $T_1$ such that base relations $\{i-1,i\}$ and $\{j,j+1\}$ do not appear in $T_1$. There exists a relation $S$ in $T_1$ and an index $i = k < j - 1$ such that $\{k,k+1\} \subseteq S$ and $\{k+1,k+2\} \subseteq S$.

Lemma 3.7
Let $D$ be an Aring of size $n$ and $D'$ a schema obtained from $D$. If $D'$ has a TP $D'$ w.r.t. $D$, then there exists a m3TP $D'_3$ of $D'$ w.r.t. $D$, and $D'_3 \leq D'$.

Proof
By induction on $n$ - the size of the Aring.

Basis
For $n = 3$, $D = \{\{1,2\},\{2,3\},\{3,1\}\}$. $D'$ must be $D \cup \{\{1,2,3\}\}$. But $D'$ itself is a m3TP, so the Lemma holds.

Induction
Assume the Lemma holds for Arings of size $\leq n$. Consider any minimal TP $D'$ which is not a 3TP for the Aring of size $n+1$ (if $D'$ is not minimal, use a minimal schema $D'_M$ obtained from $D'$). By Lemma 3.4, there is some relation $S \subseteq D'$.
such that (w.l.o.g.) $S_2 \{1,2,3\}$. Since $D'$ is minimal, by Corollary 3.1 base relations $\{1,2\}$ and $\{2,3\}$ must be children of $S$ in any qual tree for $D'$. Construct schema $D_1$ from $D'$ by replacing $S$ by $S_1=S-\{2\}$, dropping $\{1,2\}$ and $\{2,3\}$, and adding $\{1,3\}$ if $S_1 \neq \{1,3\}$. Since the only relations containing attribute 2 were $\{1,2\}$, $\{2,3\}$ and $S$, $D_1$ is a TP for an Aring of size $n$ with $\{1,3\}$ as a base relation instead of $\{1,2\}$ and $\{2,3\}$. By the induction hypothesis, there exists a m3TP $D_1\leq D_1'$ for the Aring of size $n$, and $D_1\leq D_1$. Now replace relation $\{1,3\}$ in $D_1'$ (which is a leaf by Lemma 3.3) by node $\{1,2\}$ and add leaves $\{1,2\}$ and $\{2,3\}$ as children of $\{1,2,3\}$ to produce schema $D_1'$. The following is apparent:

1. $D_1'$ is a 3-tree. (Because $D_1$ is a 3-tree.)
2. $D_1\leq D_1'$. (Because $D_1\leq D_1'$, $D_1\leq D_1$ and $\{1,2,3\} \subseteq D_1'$.)
3. $D_1'$ is TP for $D'$. (Because $D\leq D_1\leq D_1'\leq D'$ and all the base relations are leaves of $D_1'$.)

We have thus shown that the Lemma holds for Arings of size $n+1$.  

**Example 3.3**

Let $D_1'$ be as in Example 3.2. A minimal 3TP obtained from $D_1'$ is $D_1'((1,2,3),(2,3,6),(3,4,5),(3,5,6))$.  

**Lemma 3.8**

Let $D'$ be a m3TP w.r.t. Aring $D$ and let $T$ be a qual tree for $D'$. Every non-base relation in $D'$ has exactly three adjacent relations in $T$.

**Proof**

By minimality no non-base relation may be adjacent to one or two relations. Suppose a non-base relation, w.l.o.g. $\{1,2,3\}$, is adjacent to four (or more) relations in $T$. Since $D'$ is 2-attribute connected, and since there are exactly three size 2 subsets of $\{1,2,3\}$, it follows that some attribute, say 1, appears in three neighbors of $\{1,2,3\}$. By minimality, there must be base relations containing attribute 1 in each of the subtrees rooted at these neighbors. This implies that there are three base relations containing attribute 1, a contradiction.
3.3. An Algorithm for Deciding Tree Projection Existence

**Tree Projection Existence over Arings (TPA):** Let D be an Aring of size n. Given a schema D' obtained from D, is there a TP D' from D' w.r.t. D.

In this section we give an algorithm to decide TPA in time polynomial in n and the length of D'. Let D' be obtained from an Aring D and suppose a TP D' of D' w.r.t. D need be constructed. A possible strategy is that of building tree projections for groups of overlapping base relations (i.e. relations adjacent along the Aring) using projections of relations in D'. Stretch \([i,j]\) \((j > i)\), can be spanned by D' if it is possible to construct a TP w.r.t. overlapping base relations \([i,i+1],[i+1,i+2],..., [j-1,j]\) which has a qual tree with relation \([i,j]\) at its root, using projections of relations in D'. The length of a stretch \([i,j]\) is \(j-i\). Base relations are defined as stretches \([i,i+1]\) of length 1.

**Example 3.4**

Let D be the Aring over attributes \(\{1,2,3,4,5,6\}\), and D' = D. If it is possible to construct a TP w.r.t. base relations \(\{3,4\}\), \(\{4,5\}\) and \(\{5,6\}\) using projections of relations in D' with relation \(\{3,6\}\) at its root, then in our terminology stretch \([3,6]\) can be spanned by D'.

To prove that some schema D' has a TP w.r.t. D, it suffices to show that stretch \([1,n]\) can be spanned by D'. In that case there exists a TP w.r.t. base relations \(\{1,2,3,...,n-1,n\}\) whose qual tree has relation \(\{1,n\}\) at its root. This is a TP w.r.t. D. By Lemma 3.7 one need only check for the existence of a m3TP, thus it suffices to check whether stretch \([1,n]\) can be spanned, using only three attribute projections of relations in D'.

Algorithm CheckA, based on these ideas, is presented below; when given the size n of an Aring and a schema D' obtained from that Aring, the algorithm repeatedly produces stretches that can be spanned by D'. New stretches are taken as building blocks for additional stretches. The algorithm terminates when no new stretches can be produced. If stretch \([1,n]\) is produced, it follows that it can be spanned and the algorithm returns true; otherwise false is returned. A detailed
description of the algorithm is given in the Appendix. The algorithm's correctness is proved in Lemmas 3.9 and 3.10, and Theorem 3.1.

Example 3.5

Let D and D' be as in Example 3.4. Suppose stretch [3,6] can be spanned by D'. If there exists a relation S ≤ D' such that S ⊆ {2, 3, 6}, then stretch [2, 6] can be spanned by D'. The TP for stretch [2, 6] has relation {2, 6} as its root. The root is connected to relation {2, 3, 6} whose descendants are base relation {2, 3} and relation {3, 6}.

function CheckA(n:integer; D':schema):boolean; /* n is size of Aring */
var currentS : set_of_stretches; /* all stretches produced thus far */
triples : set_of_triples; /* set of three attribute subsets of relations */
startpoints : array[1..n] of points; /* points is a set-of-integers */
/* startpoints[i] contains those j for which \exists stretch [j,i] ∈ currentS */
endpoints : array[1..n] of points; /* endpoints[i] contains those j for which \exists stretch [i,j] ∈ currentS */
Sq : queue_of_stretches; /* holds all stretches to be checked for extensions */

procedure ExtendS(str:stretch;.../
var Sq:queue_of_stretches; var currentS:set_of_stretches);
/* tries to create additional stretches from a new stretch */
begin let str = [i,j];
for all k ∈ startpoints[i] do /* try to extend stretch to lower attributes */
if [k,i] ∈ currentS /* this is potentially a new stretch */
then begin /* extend */
update startpoints[j] and endpoints[k];
Sq:=addq(Sq,[k,j]); /* record a new stretch */
currentS:=currentS∪[k,j]; /* record this stretch */
end;
for all k ∈ endpoints[j] do /* try to extend stretch to higher attributes */
similar to extension into lower attributes;
end;

begin /* main */
put all three attribute subsets of relations in D' into triples;
currentS := φ; Sq:=an_empty_queue /* initialize */
create all stretches of length 1, i.e. all stretches of the form [i,i+1],
except [n,1], and add them to currentS and Sq.
 Appropriately initialize arrays startpoints and endpoints;
repeat /* produce all possible stretches */
str:=frontq(Sq); /* consider a new stretch and remove it from Sq */
ExtendS(str,Sq,currentS) /* try to extend */
until emptyq(Sq);
CheckA := ([1,n]∈currentS)
end.
We give a relatively detailed example of the execution of algorithm CheckA.

Example 3.6

Let \( D \) be the Aring over attributes \{1,2,...,6\}, and \( D'=D\cup\{(3,4,5),(1,2,3,5),(1,2,3,6),(3,4,5,6)\} \). Note that \( D' \) is cyclic, but has a TP.

Set triples includes \{1,2,3\}, \{1,2,5\}, \{1,2,6\}, \{1,3,5\}, \{1,3,6\}, \{2,3,5\}, \{2,3,6\}, \{3,4,5\}, \{3,4,6\}, \{3,5,6\} and \{4,5,6\}. Set \( currentS \) and queue \( Sq \) initially hold stretches \{1,2\}, \{2,3\}, ..., \{5,6\}. Stretches of length 2 produced and added to \( currentS \) and \( Sq \) are \{1,3\}, \{3,5\} and \{4,6\}. Next, stretch \{3,6\} (using triple \{3,5,6\}), stretch \{1,5\} (using triple \{1,3,5\}) and stretch \{2,5\} (using triple \{2,3,5\}) are produced. The only additional extension possible produces stretch \{1,6\} from stretches \{1,3\} and \{3,6\} using triple \{1,3,6\}. As stretch \{1,6\} is produced, algorithm CheckA returns true.

Lemma 3.9

Let \( D \) be the Aring over attributes \{1,2,...,n\}, and \( D'\supseteq D \). If there exists a TP of \( D' \) w.r.t. \( D \), then algorithm CheckA returns true on input \( D' \) and \( n \).

Proof

By Lemma 3.7, \( D' \) has a m3TP \( D'_1 \). Let \( T \) be a quail tree for \( D'_1 \). Root \( T \) at relation \{1,n\} and traverse it in postorder. We show that every node of \( T \) visited during postorder traversal corresponds to a stretch which is added to \( currentS \) (and recorded in \( Sq \), startpoints and endpoints) during the execution of CheckA.

For all leaves of \( T \) (which are the base relations) stretches of length 1 are added to \( currentS \) during initialization. By Lemma 3.3, all other nodes are relations of size 3.

Claim: If node \( X=[i,j,k] \) \((i<j<k)\) is visited during postorder traversal then stretch \([i,k]\) is added to \( currentS \) during the execution of algorithm CheckA.

Proof of claim. By induction on the order in which relations of size 3 are visited during the postorder traversal of \( T \).

Basis

One node, \( X \), visited. By Lemma 3.8, \( X \) has two children. Obviously, both are base relations and \( X=[i,i+1,i+2]\subseteq D'_1 \). Therefore, for some \( S\subseteq D' \), \( [i,i+1,i+2]\subseteq S \).
Since stretches \([i,i+1]\) and \([i+1,i+2]\) are in currentS and are recorded in start-points and endpoints, it follows from examination of the algorithm that CheckA adds stretch \([i,i+2]\) to currentS during its execution.

**Induction**

Assume that the claim holds for all nodes of size 3 visited so far during the traversal. Next, visit node \(X\); there are three cases to consider.

Case (1). If \(X = \{i,i+1,i+2\}\) then we claim that \([i,i+1]\) and \([i+1,i+2]\) must be adjacent to \([i,i+1,i+2]\) in \(T\). If not, disconnect them from their parents and attach them directly to \([i,i+1,i+2]\). Now, attribute 2 appears in other relations where in fact it is not needed; a violation of minimality. So, the proof of case (1) is identical to that in the basis.

Case (2). If \(X = \{i,i+1,k\}\) with \(k > i+2\) then, by an argument similar to that of case (1), one child of node \(X\) is \([i,i+1]\). The other child, call it \(S\), is \([i+1,m,k]\) for some \(i+1 < m < k\). In proof, suppose not. By Lemma 3.2, \(|S \cap X| = 2\). If \([i,i+1] \in S\), this contradicts the minimality of \(D'\). If \([i,k] \in S\), then because of minimality, the subtree of \(T\) rooted at \(S\) must include relation \([1-i,1]\). (We know it does not include \([i,i+1]\).) By Lemma 3.1 the relations on the (unique) path from relation \([i-1,1]\) to the root relation \([1,n]\) must all have some attribute \(1 \leq p \leq i-1\). Since \(T\) is rooted at relation \([1,n]\), and relation \([i-1,1]\) is a descendant of relation \([i,i+1,k]\), relation \([i,i+1,k]\) must also include such an attribute \(p\), a contradiction. Therefore, for some \(n\), \(S \not\subseteq [i+1,m,k]\). If \(m > k\) or \(m < i+1\), by using Lemma 3.1 again a contradiction can be derived. So \(i+1 < m < k\) as claimed.

Since traversal is in postorder, nodes \([i,i+1]\) and \([i+1,m,k]\) have already been visited. By the induction hypothesis, stretch \([i+1,k]\) is added to currentS during execution of CheckA (and is recorded in \(S\), startpoints and endpoints).

Since the algorithm checks all possible extensions of stretch \([i+1,k]\), stretch \([i,i+1]\) is added to currentS during initialization and for some \(S \subseteq D'\), \([i,i+1,k] \subseteq S\), stretch \([i,k]\) is produced by algorithm CheckA and added to currentS and the other data structures. The same arguments hold for the
case \( X = [i, j, j+1] \) with \( j > i + 1 \), and so stretch \([i, j+1]\) is added to \(currentS\).

Case (3). If \( X = [i, j, k] \) with \( j > i + 1 \) and \( k > j + 1 \) then the children of \( X \) must be \([i, m, j] \ (i < m < j) \) and \([j, p, k] \ (i < p < k)\). (See the discussion in case (2).) Since the traversal is in postorder, both nodes have already been visited. By the induction hypothesis, stretches \([i, j]\) and \([j, k]\) are produced by the algorithm and added to \(currentS\). One of the stretches must be produced first. As the second stretch is produced all possible extensions are checked. Since for some \( S \in D' \), \([i, j, k] \subseteq S\), stretch \([i, k]\) is also produced and added to \(currentS\).

This proves the claim.

At the end of the traversal, node \([i, j, n] \ (1 < j < n)\) which is the only child of root \([1, n]\) is visited. So, stretch \([1, n]\) is produced by algorithm CheckA. Thus, when CheckA terminates, it returns \textsc{true}. \hfill \blacksquare

**Lemma 3.10**

If on input \( D' \) CheckA produces stretch \([i, j]\) then it can be spanned by \( D'\).

**Proof**

By induction on the maximum length of a stretch produced by CheckA.

**Basis**

For stretches of length 1 \((j-i=1)\), produced at initialization, \([i, j]\) is a base relation and therefore stretch \([i, j]\) can be spanned by \( D'\).

**Induction**

Assume the Lemma holds for all stretches of length \( \leq n \). Now suppose stretch \([i, j]\) of length \( n+1 \) is produced. This is possible only if there is some \( k \), \((i < k < j)\) such that \([i, k] \subseteq currentS\), \([k, j] \subseteq currentS\) and \([i, k, j] \subseteq \text{triples}\). Clearly stretches \([i, k]\) and \([k, j]\) are both of length \( \leq n \); by the induction hypothesis they can be spanned and there exists a TP w.r.t. base relations \([i, i+1], \ldots, [k-1, k]\) with relation \([i, k]\) as its root, and a TP w.r.t. base relations \([k, k+1], \ldots, [j-1, j]\) with relation \([k, j]\) as its root. The tree illustrated in Figure 3.3 is a TP w.r.t. base relations \([i, i+1], \ldots, [j-1, j]\). Therefore stretch \([i, j]\) can be spanned by \( D'\). \hfill \blacksquare
Theorem 3.1
Algorithm CheckA returns true on input $n$ and schema $D'$ iff there exists a TP from $D'$ w.r.t. the ‘Aring over attributes $\{1,2,\ldots,n\}$.

Proof

(←) This is Lemma 3.9.

(→) If CheckA returns true, then stretch $[1,n]$ is produced. By Lemma 3.10 stretch $[1,n]$ can be spanned, so there exists a TP w.r.t. base relations $\{1,2\}, \{2,3\}, \ldots, \{n-1,n\}$ and a qual tree for it with $\{1,n\}$ as its root, built with relations that are projections of relations in $D'$. This TP is a TP of $D'$ w.r.t. the ‘Aring over $\{1,2,\ldots,n\}$.

Remark By recording for each new stretch the two stretches which produced it a TP may easily be obtained (if one exists).

Example 3.7
Consider $D$ and $D'$ from Example 3.6. Stretch $[1,6]$ is produced using triple $\{1,3,6\}$ and stretches $[1,3]$ and $[3,6]$. Stretch $[1,3]$ is produced from stretches $[1,2]$ and $[2,3]$, using triple $\{1,2,3\}$. Stretch $[3,6]$ is produced using triple $\{3,5,6\}$ and stretches $[3,5]$ and $[5,6]$. Stretch $[3,5]$ is produced from stretches $[3,4]$ and $[4,5]$, using triple $\{3,4,5\}$. The triples used in the execution of the algorithm to produce stretch $[1,6]$, together with the base relations are a TP of $D'$ w.r.t. $D$: $D''=D\cup\{\{1,2,3\},\{1,3,6\},\{3,4,5\},\{3,5,6\}\}$. Note that $D''$ is a m3TP.
Let \( w \) be the number of relations in \( D \). Assume that \( D \) is given as a list of relations with the attributes in each relation listed in sorted order. By choosing appropriate data structures we can show the following.

**Lemma 3.11** (see Appendix)

Algorithm CheckA terminates in time \( O(|D|^3 + wn^3) \).
4. QUERIES OVER DIAMONDS

4.1. DIAMONDS

We define a new class of simple cyclic schemas called Diamonds. A Diamond schema is isomorphic to the schema containing the following relations:

\[ \{A_1, B_1, C_1\} \] (junction),
\[ \{A_{n1}, B_{n2}, C_{n3}\} \] (junction),
\[ \{A_1, A_2\}, \{A_2, A_3\}, \ldots, \{A_{n1-1}, A_{n1}\} \] (bridge),
\[ \{B_1, B_2\}, \{B_2, B_3\}, \ldots, \{B_{n2-1}, B_{n2}\} \] (bridge),
\[ \{C_1, C_2\}, \{C_2, C_3\}, \ldots, \{C_{n3-1}, C_{n3}\} \] (bridge),

for any three positive integers \( n_1 \geq 1, n_2, n_3 \geq 2 \). These three integers define the Diamond. The minimal (least number of edges) qual graph for a Diamond is given in Figure 4.1.

The two relations having three attributes are called junctions; and a sequence of relations connecting the junctions is called a bridge. The length of a bridge is the number of relations in it. If \( n_1 = 1 \) then one bridge is of length zero, and the junctions are \( \{A, B_1, C_1\} \) and \( \{A, B_{n2}, C_{n3}\} \). Basically, a Diamond is a superposition of three Arings. Each Aring is identified by the junctions and two (out of three) bridges.

**Example 4.1**

The schema \( D = \{(1,2), (2,3), (4,5), (5,6), (6,7), (1,7,9), (3,4,8)\} \) is a Diamond schema with \( n_1 = 3, n_2 = 4 \) and \( n_3 = 2 \). The junctions are \( \{1, 7, 9\} \) and \( \{3, 4, 8\} \), the bridges

\[
\begin{align*}
\{B_1, B_2\} & \rightarrow \cdots \rightarrow \{B_{n_2-1}, B_{n_2}\} \\
\{A_1, B_1, C_1\} & \rightarrow \{A_1, A_2\} \rightarrow \cdots \rightarrow \{A_{n_1-1}, A_{n_1}\} \rightarrow \{A_{n_1}, B_{n_2}, C_{n_3}\} \\
\{C_1, C_2\} & \rightarrow \cdots \rightarrow \{C_{n_3-1}, C_{n_3}\}
\end{align*}
\]

Figure 4.1
are of length 2, 3 and 1. Figure 4.2 shows the minimal qual graph for \( D \). ■

4.2. Properties of Tree Projections Over Diamonds

A 4-tree schema is a tree schema in which no relation has more than four attributes. Let 4TP (m4TP) denote a TP which is a also a 4-tree schema (respectively a minimal 4TP). A schema \( D \) is 3-attribute connected if in all qual graphs for \( D \), the cardinality of the intersection of any two adjacent relations is at most 3.

Lemma 4.1

Let \( D \) be a Diamond and \( X \subseteq \cup(D) \) be the attributes appearing in one of the Arings of \( D \). (W.l.o.g. \( X = \bigcup_{i=1}^{m} A_{i} \bigcup_{i=1}^{n} B_{i} \)) The Aring is \( D_{A} = \text{rst}(D,X) \). Let \( D' \) be a schema obtained from \( D \), \( D' \) a TP of \( D' \) w.r.t. \( D \), and \( D'A = \text{rst}(D',X) \). Then \( D'A = \text{rst}(D',X) \) is a TP of \( D'A \) w.r.t. \( D' \).

Proof

By elementary properties of the rst-operation, for any two schemas \( D_{1} \) and \( D_{2} \) we have (*) \( D_{1} \leq D_{2} \Rightarrow \text{rst}(D_{1},X) \leq \text{rst}(D_{2},X) \) for a set of attributes \( X \). Since \( D \leq D' \leq D' \), property (*) implies that \( D_{A} \leq D'A \leq D'A \). Also, \( D'A \) is a tree schema since \( D' \) is a tree schema. ■

Lemma 4.2

If \( D' \) is a m4TP w.r.t. a Diamond schema then \( D' \) is 3-attribute connected.

\[
\begin{align*}
\{1,2\} & \quad \{2,3\} \\
\{1,7,8\} & \quad \{8,9\} \quad \{3,4,9\} \\
\{6,7\} & \quad \{4,5\} \\
\{5,6\}
\end{align*}
\]

Figure 4.2
Proof

Let \( T \) be a qual tree for \( D' \). Since \( D' \) is a 4-tree schema, the intersection of any two relations is of cardinality \( \leq 4 \). Suppose it is exactly 4. This implies that both relations are identical, contradicting the fact that \( D \) is minimal. \( \square \)

**Lemma 4.3**

Let \( D' \) be a minimal TP w.r.t. a Diamond schema \( D \). Let \( T \) be a qual tree for \( D' \). Then the intersection of any two relations adjacent in \( T \) is of cardinality \( \geq 2 \).

**Proof**

Let \( S_1 \) and \( S_2 \) be two relations adjacent in \( T \). Root \( T \) at relation \( S_2 \). Let \( T_1 \) be the subtree of \( T \) rooted at relation \( S_1 \), and let \( T_2 \) be the tree obtained from \( T \) by disconnecting relation \( S_1 \) (and its subtree). By minimality of \( D' \), \( T_1 \) (resp. \( T_2 \)) must include at least one base relation \( R_i \) (resp. \( R_j \)). Let \( X \) be the attributes appearing in one of the Arings of \( D \), such that \( R_i \cap X \neq \emptyset \) and \( R_j \cap X \neq \emptyset \). Let \( D_A = \text{rst}(D,X) \) and \( D''_A = \text{rst}(D',X) \) (as in Lemma 4.1). \( R_i \cap X \) and \( R_j \cap X \) are base relations of the Aring \( D_A \). \( S_1 \cap X \) and \( S_2 \cap X \) are relations in \( D''_A \), which by Lemma 4.1 is a TP w.r.t. the Aring. Lemma 3.1 implies that \( D''_A \) is 2-attribute connected, and \(|S_1 \cap S_2| \geq 2 \) follows. (Notice that the TP produced by restriction is not necessarily minimal. However, minimality was used in the proof of Lemma 3.1 only to assure that every subtree of a tree projection contains some but not all base relations. In the proof of this Lemma we know that \( R_i \) and \( R_j \) do exist.) \( \square \)

Let \( D' \) be a minimal TP w.r.t. a Diamond schema \( D \), and \( T \) a qual tree for \( D' \). W.l.o.g. \( R_i \) and \( R_e \) are the junctions of \( D \) and \( R_i = S_0, S_1, \ldots, S_e, S_{e+1} = R_e \) are the relations in \( D' \) on the path in \( T \) from \( R_i \) to \( R_e \).

**Lemma 4.4**

For \( 1 \leq i \leq k \), \(|S_i| \geq 4 \).

**Proof**

For \( 1 \leq i \leq k+1 \), let \( S_{i-1} \) and \( S_i \) be relations. \( S_1 \) and \( S_2 \) in the proof of Lemma 4.3, and use junctions \( R_i \) and \( R_e \) for relations \( R_i \) and \( R_j \). We may now choose as \( X \) the attributes of any one of the three Arings of \( D \). It follows that \(|S_{i-1} \cap S_i| \geq 3 \), since
there are no two attributes in $\mathcal{U}(D)$ appearing in all three Arings. By minimality of $D''$, it follows that for $1 \leq i \leq k$, $|S_i| \geq 4$. (Only base relations may be subsets of other relations.)

**Lemma 4.5**

Let $D'$ be a minimal TP w.r.t. a Diamond schema and let $T$ be a qual tree for $D'$. The leaves of $T$ are exactly the base relations of the Diamond.

**Proof**

All leaves must be base relations because of the minimality of $D'$. All base relations of size 2 are leaves (see the proof of Lemma 3.3). Suppose, by way of deriving a contradiction, that junction $R_i$ is not a leaf in $T$. W.l.o.g. $R_i = \{A_1, B_1, C_1\}$. Let $S_1$ and $S_2$ be two relations adjacent to $R_i$ in $T$ and suppose, w.l.o.g., that $S_1$ is on the path connecting the junctions. By Lemma 4.3, $|R_i \cap S_2| \geq 2$. Root $T$ at $R_i$. $S_2$ is the root for a subtree $T_1$ of $T$. W.l.o.g. let $A_1 \in R_i \cap S_2$, and let $A_y$ be an attribute appearing in $T_1$ which occurs the farthest away on the same bridge as $A_1$. $A_y$ exists and it is not $A_1$, otherwise $A_1$ is clearly non-essential in $S_2$. Notice that if $D$ has a bridge of length 0, then $A_1$ is not the attribute of that bridge, since otherwise it is clearly non-essential in $S_2$. Let $A_{n_1}$ be the attribute in $R_0$ on the same bridge as $A_1$ and $A_y$. Clearly, $A_y$ is not $A_{n_1}$, otherwise the tree would not be $A_{n_1}$-connected. (The other junction is not in $T_1$, and $A_{n_1} \in R_i$). Base relation $\{A_y, A_{y+1}\}$ can not be in $T_1$, because of the choice of $A_y$. It follows that $A_y \in R_i$, which is impossible. $

**Lemma 4.6**

Let $D$ be a Diamond schema, $D''$ a TP w.r.t. $D$, $T$ a qual tree for $D''$ rooted at some relation and $T_1$ a subtree of $T$. Let $G$ be a sequence of overlapping base relations $R_e, R_{e+1}, \ldots, R_f$ from some bridge (i.e. the sequence does not include any junctions) appearing in $T_1$ such that $R_{e-1}$ and $R_{f+1}$ do not appear in $T_1$. Then there exists a relation $S$ in $T_1$ and index $i \leq k < j$ such that $R_e \sqsubseteq S$ and $R_{e+1} \sqsubseteq S$. If $D''$ is minimal then $R_e$ and $R_{e+1}$ are adjacent to $S$ in $T$.

**Proof**

W.l.o.g. let $R_i = \{A_i, A_{i+1}\}$, $R_{i+1} = \{A_{i+1}, A_{i+2}\}, \ldots$. Let $S_1$ be the root of $T_1$. By attribute connectivity $S_1 \sqsubseteq \{A_i, A_y\}$. Let $X$ be the attributes appearing in one of the
Arings of $D$ that includes the bridge with the relations of $G$. The Aring is
$D_A = \text{rst}(D, X)$. $D_A' = \text{rst}(D', X)$ is a TP of $D_A'$ w.r.t. $D_A$ by Lemma 4.1. Note that since
the relations of $G$ are all on one bridge, they are left unchanged by the rst operation. By applying
Lemma 3.6 it follows that there exists a relation $S_A \in D_A'$ and
index $i \leq k < j$ such that $R_k \subseteq S_A$ and $R_{k+1} \subseteq S_A$. Since $D_A' \leq D'$, there exists a relation
$S \in D'$ such that $S_A \subseteq S$. Relations $R_k$ and $R_{k+1}$ must be adjacent to $S$ in $T$ by
minimality of $D'$.  

Lemma 4.7

Let $D$ be a Diamond schema and $D'$ a minimal TP w.r.t. $D$. Then there exist two
overlapping relations $R_k, R_{k+1} \in D'$ and a relation $S \in D'$ such that $R_k \subseteq S$ and $R_{k+1} \subseteq S$.
If $T$ is a qual tree for $D'$ then $R_k$ and $R_{k+1}$ are adjacent to $S$ in $T$.

Proof

Consider junction $R_1 = \{A_1, B_1, C_1\}$, which (by Lemma 4.5) is a leaf of $T$. Root $T$
at relation $S_1$, the (only) relation adjacent to $R_1$ in $T$. $S_1 \supset \{A_1, B_1, C_1\}$. Since $D'$
is minimal, it follows that $|S_1| \geq 4$ and it has at least three adjacent relations, each at
the root of a subtree of $S_1$ (See Figure E2). One subtree is composed of $R_1$ alone;
another, rooted at relation $S_2$, must include the second junction, $R_2$. At least one
of the attributes $A_1, B_1$ or $C_1$ must appear in one of the subtrees including neither
junction. (If not, $S_3 \supset \{A_1, B_1, C_1\}$; detach $R_1$ from $S_1$ and attach it to $S_2$; the attributes $A_1, B_1$ and $C_1$ would then be non-essential in $S_1$, contradicting the minimality of $D'$.) W.l.o.g let that attribute be $A_1$, and call the subtree of $S_1$ in which $A_1$
appears $T_1$. $T_1$ includes a sequence $G$ of base relations $\{A_1, A_2\}, \{A_2, A_3\}, \ldots,
\{A_{j-1}, A_j\}$, $2 \leq j \leq n$, such that $\{A_j, A_{j+1}\}$ does not appear in $T_1$. Since $\{A_{n_1}, B_{n_2}, C_{n_3}\}$ is
known not to be in $T_1$, clearly $j \leq n_1$. There are two cases.

Case (1). $j = 2$. Let $R_e = \{A_1, B_1, C_1\}$, $R_{e+1} = \{A_1, A_2\}$ and $S = S_1$. Obviously $A_2 \in S$, and relation
$\{A_1, A_2\}$ is adjacent to $S$ by minimality.

Case (2). $j > 2$. Then the Lemma follows immediately from Lemma 4.8.

Notice that if $D$ has a zero-length bridge then attribute $A_1$ used above cannot be
the attribute of the zero-length bridge.  


Corollary 4.1

Let $D$ be a Diamond schema with one bridge of length zero, one bridge of length one and the third bridge of length $> 1$, and let $D'$ be a minimal TP w.r.t. $D$. There exist relations $R_{x}$, $R_{x+1}$ and $S$ with the properties claimed in Lemma 4.7 such that at least one of relations $R_{x}$ and $R_{x+1}$ is part of the bridge of length $> 1$.

Proof

The junctions of the Diamond are $R_1 = \{A, B_1, C_1\}$ and $R_2 = \{A, B_2, C_3\}$; they are leaves of $T$ by Lemma 4.5. Let $S_1$ (resp. $S_2$) be the relation adjacent in $T$ to $R_1$ (resp. $R_2$). Consider two cases:

Case (1). $S_1 = S_2$ (see Figure E3). Root $T$ at relation $S_1$; it has at least four subtrees. Two subtrees consist solely of junctions, a third is base relation $\{B_1, B_2\}$ since $\{B_1, B_2\} \subseteq S_1$, and by minimality of $D'$. The other subtrees are tree projections w.r.t. base relations from the C-bridge (the bridge with C attributes). Consider the subtree $T_1$ including base relation $\{C_1, C_2\}$. If this is the only base relation in $T_1$ then $S_1 \supseteq \{C_1, C_2\}$, and by minimality $\{C_1, C_2\}$ is adjacent to $S_1$. Set $R_x = R_1$, $R_{x+1} = \{C_1, C_2\}$ and $S = S_1$ to satisfy the Lemma. Otherwise, $T_1$ includes a sequence $G$ of overlapping base relations $\{C_1, C_2\}, \{C_2, C_3\}, \ldots, \{C_{j-1}, C_j\}, 2 < j \leq 3$. Now the proof follows directly from Lemma 4.6.

Case (2). $S_1 \neq S_2$ (see Figure E4). As attribute $A$ only appears on the path connecting $R_1$ and $R_2$, the attribute $A_1$ used in the proof of Lemma 4.7 must be either $B_1$ or $C_1$. If it is $C_1$ continue as in the proof of that Lemma; clearly, $R_x$ and $R_{x+1}$ thus obtained are not on the B-bridge. Otherwise, if the attribute used is $B_1$ then relation $\{B_1, B_2\}$ is adjacent to $S_1$. This relation cannot therefore be adjacent to $S_2$. Now apply Lemma 4.7 with junction $R_2$ substituting for junction $R_1$, attribute $A_1$ chosen to be $C_{n_3}$ and the sequence $G$ being $\{C_{n_3}, C_{n_3-1}\}, \{C_{n_3-1}, C_{n_3-2}\}, \ldots, \{C_{n_3-j+1}, C_{n_3-j}\}$. The Lemma then yields the desired relations $R_x$, $R_{x+1}$ and $S$ (See Figure E5.)

Lemma 4.8

Let $D$ be a Diamond schéma and $D'$ a schema obtained from $D$. If $D'$ has a TP $D''$ w.r.t. $D$ then there exists a m4TP $D'_t$ of $D'$ w.r.t. $D$, and $D'_t \subseteq D''$. 
Proof

The proof is by induction on the total length of bridges in the Diamond.

Basis

Let $D$ be the Diamond schema shown in Figure 4.3.

In this schema the total length of bridges is 2. The possible minimal tree projections for any schema w.r.t $D$ are isomorphic to:

- $D'_1 = D \cup \{1,2,3,5\}, \{1,3,4,5\}$
- $D'_2 = D \cup \{1,2,3,4,5\}$

Observe that $D'_1$ is a m4TP and has the claimed properties. For $D'_2$ take $D'_1$ as m4TP ($D'_1 \leq D'_2$). See Figure 4.4.

Induction

Assume the Lemma holds for Diamonds whose bridges are of total length $\leq n$.

Let $D$ be a Diamond with bridges of total length $n+1$. Consider any minimal TP

---

**Figure 4.3**

---

**Figure 4.4**
D' which is not a 4-tree schema. By Lemma 4.7, there are relations ScD', Rk
and Rk+1 such that Rk, CsS and Rk+1, CsS. If D' has a bridge of length zero and a
bridge of length one then choose relation S such that one of Rk and Rk+1 is on
the bridge of length > 1. Such relations exist by Corollary 4.1. Let T be a qual
tree for D' rooted at relation S. Since D' is minimal, Rk and Rk+1 are both chil-
dren of S in T. W.l.o.g. let A_k=1, A_k+1=2, A_k+2=3 and S,= {1,2,3}. Depending on
the type of the two relations, consider two cases: (1) Neither Rk nor Rk+1 is a
junction, (2) one of Rk, Rk+1 is a junction.

Case (1). Neither relation is a junction; therefore Rk = {1,2} and Rk+1 = {2,3}.
Construct schema D_1 from D' by replacing S by S_1=S-{2}, dropping {1,2} and
{2,3}, and adding {1,3} if S_1 $\neq$ {1,3}. Obviously, D_1 is a TP w.r.t. a Diamond hav-
ing a total length of bridges ≤ n. By the induction hypothesis, there exists a
m4TP D_1t for that Diamond, and D_1t $\leq$ D_1. Let T_1 be a qual tree for D_1t.
Replace relation {1,3} in D_1t by relation {1,2,3} and add relations {1,2} and
{2,3} to obtain schema D'_{1t}. The following holds:

(1) D'_{1t} is a tree schema. (Produce T_2 by replacing leaf {1,3} in T_1 by node
{1,2,3} and adding leaves {1,2} and {2,3} as children of {1,2,3}. T_2 is a qual
tree for D'_{1t}.)

(2) D'_{1t} is a 4-tree schema. (Because D_1t is a 4-tree schema and relations
with two and three attributes were added.)

(3) D'_{1t} $\leq$ D'. (Because D_1t $\leq$ D', D_1t $\leq$ D_1 and {1,2,3} $\subseteq$ ScD'.)

(4) D'_{1t} is TP for D'. (Because D $\leq$ D' $\leq$ D' is D' is D'.)

Case (2). One of the two relations, w.l.o.g. R_k, is a junction. Then R_k = {1,2,A},
R_k+1 = {2,3} and S, = {1,2,3,A}. Construct schema D_1 from D' by replacing S by
S_1=S-{2}, dropping {1,2} and {2,3}, and adding {1,3,A} if S_1 $\neq$ {1,3,A}. Obviously,
D_1 is a TP for a Diamond having a total length of bridges ≤ n. By the induction
hypothesis, there exists a m4TP D_1t for that Diamond, and D_1t $\leq$ D_1. Now
replace relation {1,3,A} in D_1t by relation {1,2,3,A} and add relations {1,2,A}
and {2,3} to produce schema D'_{1t}. The arguments used in case (1) yield:

(1) D'_{1t} is a 4-tree schema.
Lemma 4.9

Let $D$ be a Diamond schema and $D'$ a schema obtained from $D$. If $D'$ has a TP $D'$ w.r.t. $D$ then there exists a m4TP $D_{m4}$ of $D'$ w.r.t. $D$ such that for every qual tree $T$ for $D_{m4}$ the following properties hold:

1. All relations on the path in $T$ joining the junctions (which are of size 4) include attributes from all three bridges.
2. All non-base relations which are not on the path in $T$ connecting the junctions are of size 3 and contain attributes from only one bridge.
3. All relations $S_i$ on the path in $T$ connecting the junctions are adjacent to exactly three relations. If $S_i$ is not adjacent to a junction, then it has two neighbors of size 4. If it is adjacent to a junction, $S_i$ has a neighbor of size 4 and a neighbor of size 3 (the junction). In both cases the third relation is either of size 2 (i.e. a base relation), or of size 3.

Proof

(1) By Lemma 4.8 there exists a m4TP $D'_{m4}$ of $D'$ w.r.t. $D$. Let $T_1$ be a qual tree for $D'_{m4}$, let $R_1$ and $R_2$ be the junctions and $R_1 = S_0, S_1, \ldots, S_k, S_{k+1} = R_2$ be the relations on the path in $T_1$ connecting $R_1$ and $R_2$. By Lemma 4.4, $|S_i| = 4$. Using the notation of that Lemma, by applying Lemma 3.1 to $R_1 \cap \mathbf{X}$ and $R_2 \cap \mathbf{X}$, which are two base relations of Aring $D_4$, it follows that relations $S_i$ include attributes from two bridges. As we can choose any one of the three Arings of the Diamond, it follows that relations $S_i$ must include attributes from all three bridges.

(2) Let $D'_{t}$ be a m4TP w.r.t. $D$, and $T_1$ a qual tree for $D'_{t}$. Consider the relations on the path in $T_1$ connecting the junctions. For $1 \leq i \leq k$, let $S_i = \{A_i, B_i, C_i, D_i\}$, where $D_i$ is an attribute on some bridge. Root $T_1$ at $S_1$. Since $D'_{t}$ is minimal, $S_i$ has at least three adjacent relations, each at the root of a subtree. Let $T_2$ be a subtree of $S_i$ including neither junction. (See Figure E6.) By minimality, $T_2$ includes at least one attribute from $S_i$. w.l.o.g. let it be from the bridge with "A" attributes (i.e. it is either $A_i$ or $D_i$). Therefore $T_2$ must include some
base relations from that bridge. Let \( G \) be a sequence of base relations \( \{A_r, A_{r+1}\}, \{A_{r+1}, A_{r+2}\}, \ldots, \{A_{n-1}, A_n\} \) \((1 \leq r \leq n)\), all in \( T_2 \), such that \( \{A_r, A_r\} \) and \( \{A_r, A_{r+1}\} \) are not in \( T_2 \). By attribute connectivity of \( T_1 \) it follows that \( \{A_r, A_r\} \in S_i \), and they must be attributes \( A_i \) and \( B_q \). Notice that \( G \) includes base relations from only one bridge, since the junctions are known not to be in \( T_2 \). Also, there can be no additional base relations in \( T_2 \), nor can \( S_i \) have a fourth subtree. (Otherwise there is another sequence \( G_1 \) of base relations in \( T_2 \) or the fourth subtree. Then \( S_i \) must include two additional attributes from some bridge, clearly an impossibility.)

The relations in \( T_2 \) are therefore a TP w.r.t. the base relations in sequence \( G \), which is a subschema of an Aring. By Lemma 3.8, there exists a m3TP for the sequence \( G \), which is a projection of the relations in \( T_2 \). Modify \( D' \) by replacing the relations in \( T_2 \) with the relations in the m3TP. In the qual tree, for the modified schema, \( S_i \) has again three adjacent relations. Let \( T_2 \) be defined as before. Then clearly all non-base relations in \( T_2 \) are now of size 3 and include attributes from only one bridge.

After modifying the subtrees of all relations \( S_i \) in this manner, all non-base relations except the \( S_i \)'s are of size 3. If \( D \) has a length zero bridge, all relations \( S_i \) include attribute \( A_r \), which does not appear in any other non-base relations. In this case, attribute \( B_q \) is either from the B-bridge or the C-bridge, and the construction stays the same. For a bridge of length 1 there will be a tree \( T_2 \) including only the single relation of that bridge.

(3) It was shown in part (2) of the proof that each relation \( S_i \) has three adjacent relations. One relation is the root of \( T_2 \), which is of size 3, or of size 2. In the latter case \( T_2 \) consists only of that single relation. The other two relations are on the paths connecting \( S_i \) to either junction, and thus are of size 4, except when \( S_i \) is adjacent to a junction. (\( S_i \) can not be adjacent to both junctions.)

**Example 4.2**

Consider the Diamond schema \( D = \{1,2,3\}, \{2,3,4\}, \{4,5\}, \{5,6\}, \{6,7\}, \{7,8\}, \{1,7,8\}, \{3,4,9\} \) from Example 4.1. The following schema is a TP w.r.t. \( D \) :
$D'' = D \cup \{ \{3,8,9\}, \{4,5,6,7\}, \{2,3,4,8,9\}, \{1,2,4,7,8,9\} \}$. A minimal TP w.r.t. $D$ obtained from $D''$ is: $D''_{\text{TP}} = D \cup \{ \{2,3,4,9\}, \{4,5,6,7\}, \{1,2,4,7,8,9\} \}$. A minimal 4TP w.r.t. $D$, obtained from $D''_{\text{TP}}$ is: $D''_{4\text{TP}} = D \cup \{ \{1,2,7,8\}, \{2,3,4,9\}, \{2,4,7,9\}, \{2,7,8,9\}, \{4,5,6,7\} \}$. The qual tree for $D''_{4\text{TP}}$ is given in Figure 4.6. Schema $D_{34}$, as in Lemma 4.9, is obtained from $D''_{4\text{TP}}$ by replacing relation $\{4,5,6,7\}$ with relations $\{4,5,6\}$ and $\{4,6,7\}$. The qual tree for schema $D_{34}$ is shown in Figure 4.7. 

---

Figure 4.6

\[
\begin{align*}
\{1,7,8\} & \rightarrow \{1,2,7,8\} \rightarrow \{2,7,8,9\} \rightarrow \{2,4,7,9\} \rightarrow \{2,3,4,9\} \rightarrow \{3,4,9\} \\
\{1,2\} & \quad \{6,9\} \quad \{4,5,6,7\} \quad \{2,3\} \\
\{4,5\} & \quad \{5,6\} \quad \{6,7\}
\end{align*}
\]

---

Figure 4.7

\[
\begin{align*}
\{1,7,8\} & \rightarrow \{1,2,7,8\} \rightarrow \{2,7,8,9\} \rightarrow \{2,4,7,9\} \rightarrow \{2,3,4,9\} \rightarrow \{3,4,9\} \\
\{1,2\} & \quad \{6,9\} \quad \{4,6,7\} \quad \{2,3\} \\
\{4,5,6\} & \quad \{6,7\} \\
\{4,5\} & \quad \{5,6\}
\end{align*}
\]
4.3. An Algorithm for Deciding Tree Projection Existence

Tree Projection Existence over Diamonds (TPD). Let D be a Diamond schema. Given a schema D' obtained from D, is there a TP D' from D' w.r.t. D?

A polynomial algorithm for deciding TP existence w.r.t. Diamonds is shown below; it is an extension of algorithm CheckA exhibited previously. Given a schema D' w.r.t. a Diamond schema, the algorithm attempts to build tree projections w.r.t. groups of overlapping relations centered around a junction by using projections of relations in D'. Fork \{A_i, B_j, C_k\} can be spanned by D' if it is possible to construct a TP w.r.t. base relations

\{A_1, B_1, C_1\} (junction),
\{A_1, A_2\}, \{A_2, A_3\}, ..., \{A_{n-1}, A_n\} (tooth),
\{B_1, B_2\}, \{B_2, B_3\}, ..., \{B_{j-1}, B_j\} (tooth),
\{C_1, C_2\}, \{C_2, C_3\}, ..., \{C_{k-1}, C_k\} (tooth),

that has a qual tree with relation \{A_i, B_j, C_k\} at its root, using relations that are projections of relations in D'. Note that junction \{A_1, B_1, C_1\} is defined as a fork with teeth of length 0.

Example 4.3

Let D be the Diamond of Example 4.1. Fork \{1, 5, 9\} can be spanned by D' if there exists a TP w.r.t. base relations \{1, 7, 8\}, \{5, 8\}, \{8, 7\} and \{8, 9\} with relation \{1, 5, 9\} at its root, using projections of relations in D'.

If fork \{A_{n1}, B_{n2}, C_{n3}\} can be spanned by D' then there exists a TP of D' w.r.t. the Diamond. By Lemma 4.8, only the existence of a m4TP need be checked for. It follows from Lemma 4.9 that tree projections for stretches on the bridges of the Diamond can first be produced using relations of size 3 (that include attributes from only one bridge), and then tree projections for forks can be produced using relations of size 4 (that include at least one attribute from every bridge). Algorithm CheckD based on these ideas is sketched below; its correctness is proved in Lemmas 4.10, 4.11, and Theorem 4.1. A detailed description of the algorithm is given in the Appendix.
function CheckD(n1, n2, n3: integer; D': schema): boolean;
/* n1, n2, n3 define the Diamond */
/* all stretches produced so far for the three bridges */
currentF': set_offorks; /* all forks produced so far */
triples: array[A..C] of set_of_triples; /* as in CheckA, for every bridge */
quads: array[A..C] of set_of_quads; /* set of four attribute subsets of relations */
/* quads[D] has two attributes from bridge D */
startpoints: array[A..C, 1..n] of points; /* as in CheckA, for every bridge */
endpoints: array[A..C, 1..n] of points; /* as in CheckA, for every bridge */
Sq: array[A..C] of queue_of_stretches; /* as in CheckA, for every bridge */
Fq: queue_offorks; /* holds all forks not yet checked for extensions */
f:fork; /* holds a fork to be extended */

procedure ExtendF(f: fork; var Fq: queue_offorks; var currentF: set_offorks);
/* tries to produce additional forks from a new fork */
begin
let the fork be \{A_B, C_k\}
for every tooth of the fork do
begin
W.l.o.g. let the tooth be on the bridge with "A" attributes
for all A_a \in endpoints[A..A] do /* try to extend tooth */
if \{A_q, B_j, C_k\} \subseteq currentF then /* this is potentially a new fork */
if \{A_q, B_j, C_k\} \subseteq quads[A] then /* extension possible */
begin
Fq:=addq(Fq, \{A_q, B_j, C_k\}); /* record new fork */
currentF:=currentF \cup \{A_q, B_j, C_k\} /* remember this fork */
end
end; /* ExtendF */
begin /* main */
put all four attribute subsets of relations in D into quads;
produce all stretches on all three bridges, as in CheckA.
Use only relations on bridges as initial stretches (i.e., no junction is a stretch);
Fq:=addq(an_empty_queue, \{A_1, B_1, C_1\}); /* The initial fork is a junction */
repeat /* produce all possible forks */
f:=frontq(Fq); /* consider a new fork and remove it from Fq */
ExtendF(f, Fq, currentF); /* try to extend */
until emptyq(Fq);
CheckD:=\{(A_1, B_n2, C_n3) \subseteq currentF\}
end.

Lemma 4.10
Consider TPD' for the Diamond D defined by n1, n2 and n3, and D'. If there exists a
TP of D' w.r.t. D then algorithm CheckD on input n1, n2, n3 and D' returns true.
Proof:
Let D_3 be a m4TP for D' w.r.t. D with properties as in Lemma 4.9, and let T be
a qual tree for D_3. Root T at junction \{A_1, B_n2, C_n3\}. We summarize the properties
of T:
- The leaves of T are exactly the base relations, except relation \{A_1, B_n2, C_n3\}.
which is the root.

- Each relation on the path in \( T \) connecting the junctions is of size 4 and it contains at least one attribute from every bridge.

- All other non-base relations have three attributes, all from the same bridge.

Now traverse \( T \) in postorder. We show that every node of \( T \) visited during postorder traversal corresponds to either a stretch on a bridge \( D \) which is added to \( \text{currentS}[D] \), or to a fork which is added to \( \text{currentF} \) during the execution of algorithm CheckD. If \( X \) is junction \( \{A_1, B_1, C_1\} \) then during initialization the fork \( \{A_1, B_1, C_1\} \) is added to \( \text{currentF} \). If \( X \) is a base relation of size 2 then stretch \( \{A_i, A_{i+1}\} \) is added to \( \text{currentS} \) during initialization. For all other nodes of \( T \) we show the following.

Claim:

1. If a non-junction relation \( X = \{D_i, D_j, D_k\} \) (\( D_i < D_j < D_k \)) is visited during postorder traversal of \( T \) then stretch \( \{D_i, D_k\} \) is added to \( \text{currentS} \). (The \( D \) attributes stands for attributes on some bridge.)

2. If node \( X = \{A_i, B_j, C_k, D_q\} \) (\( D_q \) is an attribute on some bridge, smaller than the second attribute of that bridge in \( X \)) is visited during postorder traversal then fork \( \{A_i, B_j, C_k\} \) is added to \( \text{currentF} \).

Proof of claim. By induction on the order in which relations are visited.

1. \( X = \{D_i, D_j, D_k\} \). By Lemma 4.9, all descendants of \( X \) are relations of size \( \leq 3 \), with attributes from the \( D \)-bridge. The proof is essentially that of the claim in Lemma 3.9.

2. \( X = \{A_i, B_j, C_k, D_q\} \). Assume the claim holds for all nodes visited so far in the postorder traversal. W.l.o.g. let attribute \( D_q \) be on the same bridge as attribute \( A_i \); call it now \( A_q \). Relation \( X \) has two subtrees, \( T_1 \) and \( T_2 \). (Remember that \( T \) is rooted at junction \( \{A_n, B_n, C_n\}\);) \( T_1 \) includes junction \( \{A_1, B_1, C_1\} \). \( T_2 \) is a TP w.r.t. base relations \( \{A_q, A_{q+1}\}, ..., \{A_{q-1}, A_q\} \), (recall the proof of Lemma 4.9), and has as root a relation \( \{A_q, A_q, A_i\} \) \( (A_q \leq A_i \leq A_k) \). Since

---

By \( D_i < D_k \) or "attribute \( D_i \) is smaller then attribute \( D_k \)," we mean here that attribute \( D_i \) is closer to junction \( \{A_1, B_1, C_1\} \) than attribute \( D_k \). This is meaningful only if both attributes are part of the same bridge.
the traversal is in postorder, node \( \{A_l, A_r, A_q\} \) has already been visited and so, by the induction hypothesis, \( \{A_l, A_r\} \in \text{current}S \). Consider two cases, depending on the structure of \( T_1 \).

Case (1). See Figure E7. \( T_1 \) consists solely of junction \( \{A_1, B_1, C_1\} \). Then \( X=\{A_1, A_q, B_1, C_1\} \) and \( A_q=A_1 \). Since fork \( \{A_1, B_1, C_1\} \) is added to \( \text{current}F \) during initialization, and since for some \( S \in D' \), \( \{A_1, A_q, B_1, C_1\} \subseteq S \), when fork \( \{A_1, B_1, C_1\} \) is checked for possible extensions, fork \( \{A_q, B_1, C_1\} \) is produced and added to \( \text{current}F \).

Case (2). See Figure E8. If \( T_1 \) is not identical to junction \( \{A_1, B_1, C_1\} \), we claim that the root of \( T_1 \) is a relation \( S_1=\{A_q, B_j, C_k, D_s\} \) (\( D_s \) is an attribute on one of the bridges, smaller than the second attribute of \( S_1 \) from this bridge). From the proof of Lemma 4.4, \( |S_1 \cap X|=3 \). \( S_1 \) includes attributes \( B_j \) and \( C_k \), because \( S_1 \cap \{A_q, A_l\} \) violates minimality: for the sake of deriving a contradiction suppose \( S_1 \) includes attribute \( A_q \). Then base relation \( \{A_q, A_l\} \) can not be in \( T_1 \) or \( T_2 \). Let \( Y=(\bigcup_{i=1}^{n_1} A_i) \cup (\bigcup_{i=1}^{n_2} B_i) \) be the attributes of an Aring \( D_i \) composing \( D \). Recall that \( A_q \leq A_i \). By applying Lemma 3.1 to relations \( \{A_i, B_i\} \) and \( \{A_q, A_l\} \), which are base relations of \( D_i \), it follows that relation \( X \) must include an attribute \( A_q \leq A_i \leq A_k \), a contradiction. Therefore \( S_1 \) includes attribute \( A_q \), as claimed. By a similar argument, again using Lemma 3.1, the fourth attribute of \( S_1, D_s \), must be smaller than the other attribute from the same bridge in \( S_1 \).

Since the traversal is in postorder, node \( \{A_q, B_j, C_k, D_s\} \) has already been visited and so, by the induction hypothesis, \( \{A_q, B_j, C_k\} \in \text{current}F \). Since for some \( S \in D' \), \( \{A_q, A_i, B_j, C_k\} \subseteq S \), when fork \( \{A_q, B_j, C_k\} \) is checked for extensions, fork \( \{A_q, B_j, C_k\} \) is produced and added to \( \text{current}F \).

In both cases the claim is proved.

At the end of the postorder traversal, node \( \{A_{n_1}, B_{n_2}, C_{n_3}, D_q\} \), which is the only child of root \( \{A_{n_1}, B_{n_2}, C_{n_3}\} \), is visited, hence fork \( \{A_{n_1}, B_{n_2}, C_{n_3}\} \) is produced and added to \( \text{current}F \) by algorithm CheckD. Thus, when CheckD terminates, it returns \textbf{true}. 

**Lemma 4.11**

If algorithm CheckD on input \( D' \) adds fork \( \{A_q, B_j, C_k\} \) to \( \text{current}F \), then it can be
spanned by $D'$.

**Proof**

We claim that if stretch $[D_i, D_j]$ is added to $currentS[D]$, then it can be spanned by $D'$. The proof is similar to that of Lemma 3.10, and is not repeated.

We continue by induction on the total length of teeth in fork $\{A_i, B_j, C_k\}$.

**Basis**

The fork with total length of teeth zero is $\{A_1, B_1, C_1\}$, it is added to $currentF$ during initialization. Since $\{A_1, B_1, C_1\}$ is a base relation, it can be spanned.

**Induction**

Assume the Lemma hold for all forks with teeth of total length $\leq n$. Now suppose fork $\{A_i, B_j, C_k\}$ with teeth of total length $n+1$ is added to $currentF$. W.l.o.g. suppose this fork was produced through extension of the fork containing $A_i$.

This is possible only if for some $A_q$ ($A_q < A_i$), fork $\{A_q, B_j, C_k\}$ is added to $currentF$. stretch $\{A_q, A_i\} \in currentS$ and $\{A_q, A_i, B_j, C_k\} \in quads$. (See the proof of Lemma 3.9, case(2).) Because the teeth of fork $\{A_q, B_j, C_k\}$ are of total length $\leq n$, by the induction hypothesis, it can be spanned. Therefore there exists a TP w.r.t. base relations $\{A_1, B_1, C_1\}$, $\{A_1, A_2', \ldots, A_{q-1}, A_q\}$, $\{B_1, B_2', \ldots, B_{j-1}, B_j\}$, $\{C_1, C_2', \ldots, C_{k-1}, C_k\}$, with relation $\{A_q, B_j, C_k\}$ at its root. Also, since stretch $\{A_q, A_i\}$ can be spanned there exists a TP w.r.t. base relations $\{A_q, A_{q+1}, \ldots, A_{r-1}, A_r\}$ with relation $\{A_q, A_i\}$ at its root. The TP shown in Figure 4.8, which has the claimed property, can be constructed because $\{A_q, A_i, B_j, C_k\} \in quads$. It follows that fork $\{A_i, B_j, C_k\}$ can be spanned. $
$
**Theorem 4.1**

Algorithm CheckD returns true on input $n_1, n_2, n_3$ and schema $D'$ iff there exists a TP of $D'$ w.r.t. the 'Diamond schema defined by $n_1, n_2$ and $n_3$.

**Proof**
(→) If CheckD returns true, then fork \( \{A_1, B_n, C_n\} \) is added to \( \text{currentF} \). By Lemma 4.11, there exists a TP w.r.t. all base relations of the Diamond, except junction \( \{A_1, B_n, C_n\} \), with \( \{A_1, B_n, C_n\} \) at its root, built with relations that are projections of relations in \( D' \). This is a TP of \( D' \) w.r.t. the Diamond. ■

By choosing appropriate data structures we can prove the following.

**Lemma 4.12** (see Appendix)

Let \( w \) be as in Lemma 3.11 and let \( n = \max(n_1, n_2, n_3) \). Algorithm CheckD terminates in time \( O(|D| + wn^4) \). ■
5. CONCLUSIONS

This work analyzes tree projections relative to two classes of simple cyclic schemas called Arings and Diamonds. Tree projections need be produced in solving SNJ queries using programs. It is conjectured that forming a tree projection is also necessary when a program is augmented with an unbounded number of semijoins. This has been proved for Arings and for schemas in which every pair of attributes appears in some relation [Kla].

Polynomial time algorithms are presented for discovering a tree projection, embedded in an arbitrary schema, relative to an Aring or a Diamond schema. Such algorithms may be useful in query processors for both centralized and distributed database systems.

Some open problems are as follows:
(1) The necessity of forming a tree projection for arbitrary SNJ queries even when an unbounded number of semijoins is allowed.
(2) Identifying new classes of cyclic schemas for which TP existence can be determined in polynomial time.
(3) Analyzing the complexity of identifying tree projections relative to arbitrary cyclic schemas (this problem is in $NP$ and is conjectured to be $NP$-Complete).
(4) Finding new query processing strategies using algorithms for deciding TP existence.
REFERENCES.


[LMG] Laver, K., Mendelzon, A.O. and M.H. Graham, "Functional Dependencies on Cyclic Database Schemes", in Proc. ACM SIGMOD Conference, 79-91, San
Jose, CA, May 1983.


A. Applications of Tree-Projection Algorithms in Database Systems

A.1. General Motivation

An algorithm which decides the existence of a tree projection (from schema D' w.r.t. schema D) is called a TP algorithm. TP algorithms have practical implications. We may think of schema D as the permanent (i.e. base) relations supported by a database management system. Schema D' corresponds to these base relations together with a collection of temporary relations produced by the system during query evaluation. In this section we discuss some possible applications of TP algorithms in centralized and distributed database systems. Incorporating such algorithms into a query processor may provide new options for efficiently evaluating queries. In addition, we suggest that identification of tree projections may serve as a basis for a reasonable query processing heuristic.

We begin by stating some facts about tree schemas and semijoins. For tree databases, i.e. the base relations form a tree schema, the result of a SNJ query can always be obtained by performing semijoins in "leaf to root" order, as dictated by a qual tree for the schema. (see [BC, BG]). If the query is of the form \((R_1 \times \cdots \times R_n)[X]\), where X is not a subset of any relation (called a nonproper tree query), then semijoins in leaf to root order, followed by semijoins in root to leaf order towards relations "covering" X reduce the database state as much as possible, i.e. delimiting it to only those tuples needed to generate the result. The result can then be obtained by joining all relations that cover X again in leaf to root order, keeping only "relevant" attributes. Since the database is reduced, this step will only take time polynomial in the size of the database and the result [Yan].

The effect of a semijoin into a relation is always to delete (possibly zero) tuples from that relation, i.e., the result of \(R_i \times R_j\) is a subset of \(R_i\). Therefore, it 

\(^6\) A set of relations covers X if it forms a connected subtree of the qual tree for the query, and its union contains X.
is possible to perform any number of semijoins into intermediate relations without utilizing much additional space. Semijoins may be efficiently implemented using hash tables. These implementations perform a semijoin in linear expected time in the size of the relations participating in the semijoin while utilizing limited temporary space for hash tables.

The possibility of obtaining a solution without using additional-space for temporary relations, together with efficient implementations of the semijoin operator suggest that obtaining a solution using semijoins (if possible) is in many cases preferable to performing more joins. The above discussion also applies to a cyclic schema $D$, provided a tree projection from relations $D'$ (which are the base relations together with temporary relations as defined above) w.r.t. $D$ exists. As a consequence, centralized as well as distributed database query processors may benefit from TP algorithms. The following subsections describe these ideas in greater detail.

A.2. Centralized Systems

In centralized database systems, the fact that the evaluation of a query may be completed by only using semijoins extends the space of options available to the query processor. The course of action which the query processor chooses to follow depends on the cost function and the statistical information used when comparing different options. If no statistical information is available, then a "semijoin-only" strategy can provide a worst case upper bound on the cost of completing the computation which, if reasonable, may suggest choosing it.

Example A.1: Obtaining a Bound on Additional Cost.

Consider a database consisting of the relations $AB$, $BC$, $CD$, $DE$, $EF$ and $FA$, and the query $(AB \times BC \times CD \times DE \times EF \times FA)[A]$. This query may be evaluated by computing $ABCF := BC \times AF$, $CDEF := CD \times EF$, $ABCF \times AB$, $CDEF \times DE$, $ABCF \times CDEF$ and then projecting relation $ABCF$ on attribute $A$. Execution of semijoins can only reduce the size of relations, and therefore no
additional space is needed once the two (real) joins have been computed. Assuming all six original relations are of size $m$, the time needed to obtain the solution in the way outlined above is $O(m^2 \log m)$, using $O(m^2)$ space in the worst case, even if joins and semijoins are performed in a "naive" way by sorting. A different approach might take substantially more time in the worst case. After producing relations ABCF and CDEF, a query processor might choose the following sequence of operations: $ABCF := ABCF \times AB$, $CDEF := CDEF \times DE$, $ABCDEF := ABCF \times CDEF$, and obtain the result by projecting $ABCDEF$ onto $A$. This strategy might cost $O(m^4)$ time and space.

In order for tree projections to be utilized, the existence of a TP should be checked for as often as possible, preferably every time a new temporary relation is produced. It would therefore be advantageous if the TP algorithm used could be modified to work incrementally, i.e. at every step exploring only the new possibilities made available by the additional temporary relation. Note that the TP algorithms given for Arings and Diamonds may be easily modified efficiently to work in incremental mode.

Intermediate results, obtained while evaluating queries, and saved by the system, may be used as part of a tree projection in evaluating a new query. Roughly speaking, using a projection of an existing relation is another possible "access path". Consider query $Q$ of the form $\pi_s (R_f) [X]$ where $s$ is a set of indices and $X$ a set of attributes. Any previously obtained result of the form $Q_p = \pi_{i \in s} R_f [Y]$, where $s \subseteq r$, may be used for producing the result for $Q$, provided the underlying relations have not been changed in the meantime by either adding or removing tuples. The schema of $Q_p$, i.e. $Y$, together with schemas of other previously obtained valid results will be part of schema $\mathcal{D}$ which is the input to a TP algorithm.

Example A.2: Using Intermediate Results Produced By Previous Queries.

If relations ABCF and CDEF of Example A.1 already exist in the system, the
query may be evaluated by semijoins alone, using no additional space for intermediate relations. In case relation ABCF, say, does not exist, but as part of a previous query relation \( R = (BC \times DE \times AF)[ABCDF] \) was computed, we may now obtain ABCF by computing \( R[ABCF] \). This holds only if base relations BC, DE and AF have not been changed since \( R \) was computed.

Semijoins may be especially useful if intermediate results are produced in a way facilitating efficient semijoins (similar to “reformattting” in INGRES [SWKH]). This is possible by “looking ahead” when planning a solution to a given query; i.e. by identifying the existence of a future tree projection, relations may be produced already sorted or hashed in useful ways.

The following example shows that identifying tree projections may save time in multiple query processing. (See [Jar] for a survey on this subject.) Again, look ahead can be used when solving multiple queries posed as a batch.

**Example A.3: Multiple Queries.**

Consider a database containing the five relations ABE, BC, CDF, AD, EF. Suppose the following queries are posed as a batch:

- (Q1) \( BC \times CDF \),
- (Q2) \( ABE \times AD \),
- (Q3) \( EF \times ABE[BE] \times CDF[DF] \),
- (Q4) \( (ABE \times BC \times CDF \times AD \times EF)[BDE] \).

When solving Q1, Q2, and Q3 the relation schemas BCDF, ABDE and BDEF, respectively, are created. It turns out that these relations form a tree projection w.r.t. the relations of query Q4. Therefore, semijoins from BCDF into BDEF and from ABDE into BDEF will suffice to produce the correct result in BDEF. If the existence of a tree projection for query Q4 is detected before queries Q1, Q2 and Q3 are evaluated, relation BDEF may be produced indexed or sorted on attributes BDE and BDF. Even if this is not done, using semijoins can be considered as an additional option for computing Q4. ♦
A.3. Distributed Systems

In distributed database systems, the use of TP algorithms offers advantages similar to those for centralized systems. In addition, semijoins may substantially save time because in evaluating the semijoin of \( R_i \) in site 1 and \( R_j \) in site 2, only the columns "common" to \( R_i \) and \( R_j \) need be transmitted (see [GBWR]). This especially holds in systems where the dominant cost factor is communication and local computations are considered relatively "cheap", an approach taken in SDD-1 [GBWR].

As in the centralized case, results of previous computations which are still valid may be used as part of a tree projection. If memory space at sites is considered cheap, keeping intermediate results (obtained while evaluating queries) available for later use can save communication cost, as the following example shows.

**Example A.4: Using Intermediate Locally Available Results from Previous Queries.**

Consider a distributed database system, having four sites. Suppose relations \( AB, BC, CD \) and \( DA \) reside at sites 1, 2, 3 and 4, respectively, and the query \((AB \bowtie CD \bowtie DA)\bowtie[A]\) need be evaluated. Suppose also that the following intermediate results already exist:

\[
\begin{align*}
ABD &= AB \bowtie AD \text{ at site 1,} \\
ABC &= AB \bowtie BC \text{ at site 2,} \\
ACD &= AD \bowtie CD \text{ at site 3.}
\end{align*}
\]

Since relations \( ABC \) and \( ACD \) form a tree-projection, the query may be solved by only shipping \( ACD[A] \) from site 3 to site 2. The result of the query is obtained by locally computing \((ABC \bowtie AC)[A]\) at site 2.

An additional application for TP algorithms arises in the context of distributed database systems maintaining multiple copies of relations at different sites (as in System R* [W*]). If a tree projection with respect to a query may be obtained by locally joining relations at various sites, the result of a query may be computed by
only using semijoins. Note that this is a special case of a normal-distributed database, with relations "shipped" in advance to various sites.

**Example A.5: Using Multiple Copies.**

Consider a distributed database system with three sites, maintaining the relations \( AB, CD, DE, AEF \) and \( BCF \) with multiple copies of relations distributed as follows:

- Site 1: relations \( AB, DE \) and \( AEF \).
- Site 2: relations \( AB, CD \) and \( BCF \).
- Site 3: relations \( CD, DE \) and \( BCF \).

Relations \( A\overline{BCDF} \), \( A\overline{BCDEF} \) and \( B\overline{CDEF} \) may be produced by local processing.

Suppose the join of all relations, projected on attributes \( AB \) need be computed. The schema consisting of the original relations together with relations \( A\overline{BCDF} \), \( A\overline{BCDEF} \) and \( B\overline{CDEF} \) is cyclic. However, relations \( ADEF, A\overline{BDF} \) and \( B\overline{CF} \), together with the original relations form a tree projection. The desired result may thus be obtained by computing the semijoin of \( ADEF \) into \( A\overline{BDF} \), the semijoin of \( B\overline{CF} \) into \( A\overline{BDF} \) and projecting the result on attributes \( AB \).

**B. Proof of Lemmas 3.5 and 3.6**

We now present the proofs for two Lemmas cited in the main text.

**Proof of Lemma 3.5**

By induction on \( n \), the size of the Aring.

**Basis**

For \( n=3 \), \( D=\{(1,2), (2,3), (3,1)\} \). \( D' \) must be \( D \cup \{(1,2,3)\} \). Clearly, \( k_1=1, k_2=2 \) and \( S_1=S_2=\{1,2,3\} \) satisfy the Lemma. For \( n=4, D=\{(1,2), (2,3), (3,4), (4,1)\} \). Any minimal TP w.r.t. \( D \) must be isomorphic to \( D'=D \cup \{(1,2,3), (1,3,4)\} \). (If the TP is not minimal, \( D' \) must be a projection of it.) Set \( k_1=1, S_1=\{1,2,3\}, k_2=3 \) and \( S_2=\{3,4,1\} \) to satisfy the Lemma.

**Induction**

Assume the Lemma holds for Arings of size \( \leq n \). W.l.o.g. we prove the Lemma only for minimal tree projections. (For every non-minimal schema \( D' \), there
exists a minimal schema $D'_M$ such that $D'_M \leq D''$. If the Lemma holds for $D'_M$, it clearly holds for $D''$. Consider any minimal TP $D''$ w.r.t. the Aring of size $n+1$ (if $D''$ is not minimal, use a minimal schema $D'_M$ obtained from $D''$). By Lemma 3.4, there is some relation $S \subset D'$ such that (w.l.o.g.) $S \{1,2,3\}$. Since $D'$ is minimal, by Corollary 3.1 base relations $\{1,2\}$ and $\{2,3\}$ must be children of $S$ in any qual tree for $D'$. Construct schema $D_1''$ from $D''$ by replacing $S$ by $S_1 = S - \{2\}$, dropping $\{1,2\}$ and $\{2,3\}$, and adding $\{1,3\}$ if $S \neq \{1,2,3\}$. Since the only relations in $D$ containing attribute 2 are $\{1,2\}$, $\{2,3\}$ and $S$, $D_1''$ is a TP for an Aring of size $n$ with $\{1,3\}$ as a base relation instead of $\{1,2\}$ and $\{2,3\}$. By the induction hypothesis, there exist two indices $k_1$ and $k_2$, $|k_1-k_2| > 1$, and two relations $S_{11} \subset D_1''$ and $S_{22} \subset D_1''$ such that $R_{11} \subset S_{11}$, $R_{21} \subset S_{11}$, $R_{22} \subset S_{22}$ and $R_{22} \subset S_{22}$. The only difference between schemas $D_1''$ and $D''$ is the replacement of two base relations by a single base relation. Therefore if $k_1 \neq 1$ and $k_2 \neq 1$, we can use both indices, and setting $S_1 = S_{11}$ and $S_2 = S_{22}$ will satisfy the Lemma. If one of the indices is (w.l.o.g.) $k_1 = 1$ then setting $S_1 = \{1,2,3\}$ will again satisfy the Lemma. Note that $|k_1 - k_2| > 1$ clearly holds by the induction hypothesis and since the indices are not changed.

We have thus shown that the Lemma holds for Arings of size $n+1$.

**Proof of Lemma 3.6**

W.l.o.g $D'$ is minimal. (Otherwise, replace it by a minimal TP $D'_M$ derived from $D'$ and observe that $D'_M \leq D''$.) Let $S_1$ be the root of $T_1$. If $S_1$ is also the root of $T$, $T_1$ and $T$ are identical and the Lemma simply restates Lemma 3.4. We therefore assume that $S_1$ has a parent $S_2$ in $T$. Since base relations $\{i-1, i\}$ and $\{j, j+1\}$ do not appear in $T_1$, by attribute connectivity, $S_1 \cap S_2 = \{i, j\}$.

Let $D_1'' \subset D'$ be the schema composed of all relations appearing in $T_1$. Consider two cases.

**Case (1).** The only base relations in $D_1''$ are the relations in sequence $G$. Then, by minimality of $D'$, $S_1 \cap S_2 = \{i, j\}$. If $S_1 \neq \{i, j\}$, add a new relation $\{i, j\}$; place it between $S_1$ and $S_2$ in $T$ and let $T_1$ be rooted at this new relation. $D_1''$ is now a tree projection w.r.t. relations $\{i, i+1\}, \ldots, \{i-1, j\}, \{i, j\}$ which form an Aring (of size $j-i+1$). If
j=i+2 than $D_1''$ is a TP w.r.t. an Aring of size 3, and it must include relation $\{i,i+1,j\}$ and the Lemma holds. (Set $k=i$ and $S=\{i,i+1,i+2\}$.) If $j>i+2$ then, by Lemma 3.4, there exists an index $i\leq k\leq j$ and relation $S\in D_1''$ such that $\{k,k+1\} \subseteq S$ and $\{k+1,k+2\} \subseteq S$. If $i\leq k\leq j-2$, relation $S$ and index $k$ satisfy the Lemma. If $j-1\leq k\leq j$, relation $\{i,j\}$ is not a base relation of $D''$ and index $k$ does not satisfy the Lemma. However, by Lemma 3.5, there exists another index $k_2, |k_2-k|>1$, and relation $S$ such that $\{k_2,k_2+1\} \subseteq S$ and $\{k_2+1,k_2+2\} \subseteq S$. Clearly, if $j-1\leq k\leq j$ then $1\leq k_2\leq j-2$ and the Lemma holds.

Case (2). There are base relations in $T_1$ other than the relations of $G$. $D'$ can be modified as follows. Let $X=\{i,i+1,\ldots\}$, $D_x=rst(D_1'',X)$ and $D_{x\setminus X}=del(D_1'',X)$. These two new schemas are tree schemas by properties of the rst and del operations, and since $D_1''$ is a tree schema. Replace the relations in $D_1''$ by the relations in $D_x\cup D_{x\setminus X}$. A qual tree for $(D''\setminus D_1'')\cup D_x\cup D_{x\setminus X}$ is obtained from $T$ by replacing $T_1$ with two new trees for $D_x$ and $D_{x\setminus X}$ whose roots are attached to $S_2$. We can now use the tree for $D_x$ and continue as in the proof for case (1) to obtain index $k$ and relation $S\in D_1''$, such that $\{k,k+1\} \subseteq S$ and $\{k+1,k+2\} \subseteq S$. Since $D_x\subseteq D_1''$, there exists $S\in D'$ such that $S\subseteq S'$; $S$ satisfies the Lemma.

C: Algorithm CheckA

Algorithm CheckA is given below. Sets currentS and triples are implemented as arrays of size $n^2$ and $n^3$, respectively. Therefore, testing set membership can be done in constant time. An additional data structure, which is not mentioned in the body of the paper is an array of sets called have. $j\in have[i]$ iff relation $R_j$ has attribute $i$. We use have in procedure FindTriples in producing all three attribute subsets of relations.

```plaintext
function CheckA(n:integer; D':schema):boolean;
    /* n is size of Aring */
    type stretch = array[1..2] of integer;
    triple = array[1..3] of integer;
    points = set of integer;
    set_of_stretches = array[1..n,1..n] of boolean;
    set_of_triples = array[1..n,1..n,1..n] of boolean;
    queue_of_stretches = queue of stretch;
    var currentS: set_of_stretches; /* all stretches produced so far */
    triples : set_of_triples; /* set of three attribute subsets of relations */
```
startpoints: array[1..n] of points;

for every endpoint, where do stretches start */

endpoints: array[1..n] of points;

for every startpoint, where do stretches end */

Sq : queue_of_stretches; /* holds all stretches not yet checked for extensions */

have : array[1..n] of points; /* for each attribute, which relations include it */

str : stretch;

i : integer;

procedure FindTriples(n: integer; D: schema; var triples: set_of_triples);
/* finds all three attribute subsets of relations in D */

var i, j, k: integer;

begin
for i := 1 to n - 2 do
  for j := i + 1 to n - 1 do
    for k := j + 1 to n do
      if (have[i] and have[j] and have[k]) /= 0
        then triples[i, j, k] := true /* triple {i, j, k} is included in some relation */
        else triples[i, j, k] := false;
end;
/* FindTriples */

procedure ExtendS(str: stretch; var Sq: queue_of_stretches; var currentS: set_of_stretches);
/* tries to create additional stretches from a new stretch */

var i, j, k: integer;

begin
  i := str[1];
  j := str[2];
  for all k ∈ startpoints[i] do
    if not currentS[k, i] then
      if triples[k, i, j] /* extension possible, is new stretch */
        then /* extend */
          begin /* update data structures */
            startpoints[i] := startpoints[i] ∪ k;
            endpoints[k] := endpoints[k] ∪ j;
            str[1][1] := k;
            str[1][2] := i;
            Sq := add(Sq, str[1]); /* remember this is new stretch */
            currentS[k, i] := true /* remember this stretch */
          end;
    end;
  for all k ∈ endpoints[j] do
    if not currentS[i, k] then
      if triples[i, j, k] /* extension possible, is new stretch */
        then /* extend */
          begin /* update data structures */
            startpoints[k] := startpoints[k] ∪ i;
            endpoints[i] := endpoints[i] ∪ k;
            str[1][1] := i;
            str[1][2] := k;
            Sq := add(Sq, str[1]); /* remember this is new stretch */
            currentS[i, k] := true /* do not consider stretch again */
          end;
    end;
  end;

begin /* main */
/* initialize have */
if for i := 1 to n do have[i] := ø;
if for all relations R ∈ D do
for all attributes \( j \in R \) do
  \( \text{have}[j] := \text{have}[j] \cup \{i\} \) /* attribute \( j \) appears in relation \( R_i \) */
  \( \text{FindTriples}(n,D', \text{triples}); \) /* find all three attribute subsets of relations in \( D' \) */
for \( i := 1 \) to \( n - 1 \) do
  for \( j := i + 1 \) to \( n \) do currentS\([i,j]\) := false;
  \( \text{Sq} := \text{an empty queue}; \)
for \( i := 1 \) to \( n - 1 \) do /* for each Ring relation except \{n,1\} */
  \( \text{begin} /*, form a length one stretch and add it to data structures */ \)
  \( \text{str}[1] := i; \)
  \( \text{str}[2] := i + 1; \)
  \( \text{Sq} := \text{addq}(\text{Sq}, \text{str}); \)
  \( \text{startpoints}[i + 1] := \{i\}; \)
  \( \text{Endpoints}[i] := \{i + 1\}; \)
  \( \text{end}; \)
repeat
  \( \text{str} := \text{frontq}(\text{Sq}); /* consider a new stretch */ \)
  \( \text{Sq} := \text{removeq}(\text{Sq}); \)
  \( \text{ExtendS}(\text{str}, \text{Sq}, \text{currentS}) /* try to extend */ \)
  \( \text{until emptyq}(\text{Sq}); \)
  \( \text{CheckA} := \text{currentS}[1,n] \)
\( \text{end}. \)

\textbf{Proof of Lemma 3.11}

Initialization of \( \text{have} \) is done in one pass over \( D \), and takes \( O(|D|) \) time. Deciding whether a triple \( \{i,j,k\} \) should be in \( \text{triples} \) is done in \( O(w) \), since checking if the intersection of \( \text{have}[i] \), \( \text{have}[j] \) and \( \text{have}[k] \) is empty can be done by merging these three sets, each of which is at most of length \( w \). (Note that relations appear in all sets in the same order.) All triples are therefore found in \( O(wn^3) \) time. Initialization of \( \text{currentS} \) is done in \( O(n^2) \) time. All other initializations take \( O(n) \) time.

The \textbf{repeat} loop may be executed at most \( n^2 \) times, as there are only \( n^2 \) different stretches and care is taken not to add the same stretch twice to \( \text{Sq} \). Procedure \( \text{ExtendS} \) takes \( O(n) \) time, as there are at most \( n \) startpoints and endpoints to be considered for a given stretch. Execution of the \textbf{repeat} loop thus takes no more than \( O(n^3) \) time. So \( \text{CheckA} \) runs in \( O(|D| + wn^3) \) time. 

\textbf{D. Algorithm CheckD}

A description of algorithm \( \text{CheckD} \) follows. It is a extension of \( \text{CheckA} \), and it uses the same data structures. Some data structures are triplicated, once for every bridge. Again, all sets are implemented as arrays.

\textbf{function} \( \text{CheckD}(n1,n2,n3;D:\text{schema});\text{boolean}; \)
  /* \( n1,n2,n3 \) are parameters of the Diamond */
\textbf{let} \( n \) be \( \max(n1,n2,n3) \);
\textbf{type} \( \text{stretch} = \text{array}[1..2] \) of \( \text{integer}; \)
\textbf{fork} = \text{array}[1..3] of \( \text{integer}; \)
triple = array[1..3] of integer;
quad = array[1..4] of integer;
points = set of integer;
set_of_stretches = array[1..n,1..n] of boolean;
set_of_triples = array[1..n,1..n,1..n] of boolean;
set_of_forks = array[1..n,1..n,1..n] of boolean;
set_of_quads = array[1..n,1..n,1..n,1..n] of boolean;
queue_of_stretches = queue of stretch;
queue_of_forks = queue of fork;
name_of_bridge = A..C;

/* currentS[D] holds all stretches of fork D produced so far */
currentF: set_of_forks; /* all forks produced so far */
triples: array[A..C] of set_of_triples; /* triples[D] holds all three attribute subsets of relations such that the attributes are on bridge D */
quads: array[A..C] of set_of_quads; /* quads[D] holds all four attribute subsets of relations such that there are two attributes from bridge D */
startpoints: array[A..C,1..n] of points; /* for every endpoint on bridge D, where do stretches start */
endpoints: array[A..C,1..n] of points; /* for every startpoint on bridge D, where do stretches end */
Sq: array[A..C] of queue_of_stretches; /* Sq[D] holds all stretches on bridge D not yet checked for extensions */
Fq: queue_of_forks; /* holds all forks not yet checked for extensions */
have: array[A..C,1..n] of points; /* for each attribute on bridge D, which relations include it */
D; name_of_bridge;
str: stretch;
f: fork;
i,j,k: integer;

procedure FindTriples(D: name_of_bridge);
/* finds three attribute subsets of relations, with all attributes on one bridge */
var i,j,k: integer;
begin
  for i:=1 to n[D]-2 do
    for j:=i+1 to n[D]-1 do
      for k:=j+1 to n[D] do
        if (have[D,i] \ have[D,j] \ have[D,k]) \= \0
          then triples[D][i,j,k]:=true
          else triples[D][i,j,k]:=false
    end;
end;

procedure FindQuads(D: name_of_bridge);
/* finds four attribute subsets of relations in D' having two attributes from bridge D and one attribute from the other bridges */
var i,j,k,l: integer;
begin
  for i:=1 to n[D]-1 do
    for j:=i+1 to n[D] do
      for k:=1 to n[D@1] do /* @ means addition modulo the bridges */
        for l:=1 to n[D@2] do
          if (have[D,i] \ have[D,j] \ have[D@1,k] \ have[D@2,l]) \= \0
            then quads[D][i,j,k,l]:=true
            else quads[D][i,j,k,l]:=false
      end;
end; /* FindQuads */

procedure ExtendS(str: stretch; D: name_of_bridge;
var Sq: queue_of_stretches; var currentS: set_of_stretches);
/ * tries to create additional stretches from a new stretch */

var i,j,k : integer;
str1 : stretch;
begin
i:=str[1];
j:=str[2];
for all k ∈ startpoints[D,i] do
  if not currentS[k,j] then
    if triples[D][k,i,j] then /* extension possible, is new stretch */
      begin /* update data structures */
        startpoints[D,k] := startpoints[D,k] ∪ k;
        endpoints[D,k] := endpoints[D,k] ∪ j;
        str[1][1] := k;
        str[1][2] := j;
        Sq := addq(Sq,str1); /* remember this is new stretch */
        currentS[k,j] := true /* remember this stretch */
      end;
    for all k ∈ endpoints[D,j] do
      if not currentS[i,k] then /* extension possible, is new stretch */
        if triples[D][i,k,j] then
          begin /* update data structures */
            startpoints[D,i] := startpoints[D,i] ∪ i;
            endpoints[D,i] := endpoints[D,i] ∪ k;
            str[2][1] := i;
            str[2][2] := k;
            Sq := addq(Sq,str1); /* remember this is new stretch */
            currentS[i,k] := true /* do not consider stretch again */
          end;
end;

procedure ExtendF(fork, var Fq:queue_of_forks; var currentF:set_of_forks);
/* tries to extend fork */
var i,j,k,l : integers;
begin
i:=f[1];
j:=f[2];
k:=f[3];
for all l ∈ endpoints[A,i] do
  if not currentF[l,j,k] then
    if quads[A][l,j,k,i] then
      begin
        f[1][1] := i;
        f[1][2] := j;
        f[1][3] := k;
        Fq := addq(Fq,f1);
        currentF[l,j,k] := true
      end;
  for all l ∈ endpoints[B,j] do
    if not currentF[i,l,k] then
      if quads[B][j,l,k,i] then
        begin
          f[2][1] := i;
          f[2][2] := l;
          f[2][3] := k;
          Fq := addq(Fq,f1);
          currentF[i,l,k] := true
        end;
  for all l ∈ endpoints[C,k] do
    if not currentF[i,j,l] then

if quads[C][k,l,i,j] then
    begin
        f[1]:=i;
        f[2]:=j;
        f[3]:=l;
        Fq:=addq(Fq,f1);
        currentF[i,j]:=true
    end;
end; /* ExtendF */

begin /* main */
    /* initialize have */
    for D:=A to C do
        for i:=1 to n[D] do
            have[D,i]:=false;
    for all relations R within D do
        for all attributes Dj within R do
            have[Dj]:=false;
    Sq[D]:=an_empty_queue
end;

for i:=1 to n[A] do
    for j:=1 to n[B] do
        for k:=1 to n[C] do
            currentF[i,j]:=false;

for D:=A to C do
begin /* for each bridge relation in bridge D */
    begin /* form a length one stretch and add it to data-structure */
        str[1]:=i;
        str[2]:=i+1;
        Sq[D]:=addq(Sq[D],str);
        startpoints[D,i+1]:=str[1];
        endpoints[D,i]:=str[2];
    end;
end;
/* produce all possible stretches, the same way as in CheckA */

for D:=A to C do
    repeat
        str:=frontq(Sq[D]);
        Sq[D]:=removeq(Sq[D]);
        ExtendS(str,D,Sq[D],currentS[D]);
    until emptyq(Sq[D]);

f[1]:=1; f[2]:=1; f[3]:=1;
Fq:=addq(an_empty_queue,f);
repeat
    f:=frontq(Fq);
    Fq:=removeq(Fq);
    ExtendF(f,Fq,currentF);
until emptyq(Fq);
if currentF[n[A],n[B],n[C]]
    then return(true)
else return(false)
end.
Proof of Lemma 4.12

Initialization of have takes $O(|D|)$ time. Finding all triples and quads takes $O(w(n^3 + n^3))$ time. Initialization of currentF and currentS takes $O(n^4 + n^3)$ time. Producing all stretches takes $O(n^3)$ time, as shown in the proof of Lemma 3.11. Producing all forks takes no more than $O(n^4)$ time, because there are at most $n^3$ different forks added to $F_q$, and for each fork there are no more than $n$ possibilities for extension. Therefore CheckD runs in $O(|D| + wn^4)$ time. $\blacksquare$
Figure E1

Figure E2

Figure E3
Figure E6

Figure E7
Figure E8