Avoiding Cartesian Products for Multiple Joins

SHINICHI MORISHITA

IBM Research, Tokyo Research Laboratory, Kanagawa, Japan

Abstract. Computing the natural join of a set of relations is an important operation in relational database systems. The ordering of joins determines to a large extent the computation time of the join. Since the number of possible orderings could be very large, query optimizers first reduce the search space by using various heuristics and then try to select an optimal ordering of joins. Avoiding Cartesian products is a common heuristic for reducing the search space, but it cannot guarantee optimal ordering in its search space, because the cheapest Cartesian-product-free (CPF, for short) ordering could be significantly worse than an optimal non-CPF ordering by a factor of an arbitrarily large number. In this paper, we use *programs* consisting of joins, semijoins, and projections for computing the join of some relations, and we introduce a novel algorithm that derives programs from CPF orderings of joins. We show that there exists a CPF ordering from which our algorithm derives a program whose cost is within a constant factor of the cost of an optimal ordering. Thus, our result demonstrates the effectiveness of avoiding Cartesian products as a heuristic for restricting the search space of orderings of joins.

Categories and Subject Descriptors: F.2.2 [**Theory of Computation**]: Nonnumerical Algorithms and Problems—*sequencing and scheduling*; H.2.4 [**Information Systems**]: Systems—*query processing*; I.2.8 [**Computing Methodologies**]: Problem Solving, Control Methods, and Search—*plan execution, formation generation*

General Terms: Algorithms, Database Theory, Languages

Additional Key Words and Phrases: Cartesian product, database scheme, join expression tree, join strategy, optimality, semijoin

1. *Introduction*

Computing the natural join of a set of relations plays an important role in relational and deductive database systems. In order to solve this problem efficiently, we need to find a way of reducing the number of intermediate tuples to compute the final result. A naive strategy may produce a huge number of intermediate tuples during the computation, even though the final result is very small.

This work is based on the extended abstract of "Avoiding Cartesian products in programs for multiple joins" in *Proceedings of the 11th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (San Diego, Calif., June 2–4). ACM, New York, 1992, pp. 368–379.

Author's address: IBM Tokyo Research Laboratory, 1623-14, Shimo-tsuruma, Yamato City, Kanagawa 242, JAPAN, email: morisita@trl.ibm.co.jp.

If the scheme of these relations is acyclic, we can compute the join efficiently, namely in time polynomial in the size of the input relations and the output [Beeri et al. 1983; Bernstein and Goodman 1987; Ullman 1989]. Yannakakis [1981] has extended this idea in order to compute a project–join expression with an acyclic database scheme. For cyclic schemes, however, the problem cannot be solved in polynomial time unless P=NP, because checking whether the join of a set of relations is nonempty is an NP-complete problem [Chandra and Merlin 1977].

Thus, in general, we cannot expect to find an efficient algorithm for computing the join of some relations. Query optimizers [Selinger et al. 1989; Wong and Youseffi 1976] search the space of orderings of joins and try to select a cheap ordering that does not generate huge intermediate relations. Since the number of possible orderings could be very large, query optimizers reduce the search space by using various heuristics. One heuristic commonly used in the evaluation of joins is to use linear orderings of joins, which have the form $(\cdots(R_1 \bowtie R_2) \bowtie \cdots) \bowtie R_n$. This heuristic is of practical interest, because it allows us to keep only one temporary relation at any time. Another heuristic is to avoid Cartesian products because they tend to be expensive.

Query optimizers in many well-known systems, such as INGRES [Wong and Youseffi 1976] and System *R* [Selinger et al. 1979], use one or both heuristics. Smith and Genesereth [1985] considered linear orderings of joins (conjunctions, to them), and gave an "adjacency restriction rule" that improves the cost of a join by locally swapping two adjacent relations. Swami [1989] and Swami and Gupta [1988] used both heuristics to reduce the search space and compared several statistical techniques.

The aim of the above heuristics is to let query optimizers search a small subspace of the entire orderings of joins. However, restricting the search space by using one of the above heuristics may result in the loss of all optimal orderings from the search space; that is, both the cheapest linear ordering and the cheapest CPF ordering could be much worse than an optimal nonlinear and non-CPF ordering by a factor of an arbitrarily large number. We will give such an example in Section 2.

The following question naturally arises: Under what conditions on the actual data, can we find an optimal linear ordering or an optimal CPF ordering? Tay [1993] considers this problem and gives some sufficient conditions, which, however, are rarely expected to hold in general cases.

In this paper, we employ programs consisting of joins, semijoins, and projections for computing multiple joins, and we present a novel algorithm that derives a program from a CPF ordering of joins without looking at the actual data. Our main result is that there exists a CPF ordering from which our algorithm derives a program whose cost is within a factor k of the cost of an optimal ordering if the join of the actual data is nonempty. k is polynomial in the number of relations and independent of the amount of the actual data. In practice, the number of relations tends to be much smaller than the size of the actual data, and therefore k can be thought of as a constant.

Our result demonstrates the effectiveness of avoiding Cartesian products as a heuristic to reduce the search space of orderings of joins. Query optimizers can search the subspace of CPF orderings and try to find a CPF ordering from which an efficient program can be derived.

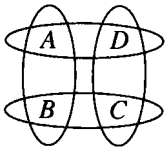


FIG. 1. Hypergraph for $\{AB, BC, CD, DA\}$.

To estimate the cost of executing an ordering of joins, instead of using a precise measure that takes into account indexes, block sizes, cache sizes, and other hardware-dependent factors, we simply use the number of tuples that appear in the input relations or the relations generated. Although not ideal, this measure is reasonable because when the cost is n, the cost of the actual best possible method is no more than $O(n \log n)$ and no less than $\Omega(n)$ [Ullman 1989].

Section 2 introduces some terms. Section 3 presents the algorithm for deriving programs from CPF orderings of joins, and then proves our main claim.

2. *Preliminaries*

2.1. RELATIONAL DATABASES. A *relation scheme* is a finite set of *attributes*. We use bold letters, such as **V**, to denote relation schemes. According to custom, ABC denotes a relation scheme that consists of the attributes A, B, and C. A *tuple* (or a *record*) over the relation scheme **V** is a mapping of the attributes in **V** to constants. For instance, (1, 2, 3) is a tuple over ABC.

A *relation* over a relation scheme **V** is a set of tuples over **V**, and is represented by $R(\mathbf{V})$. For instance, $R(AB)$ is a relation over AB. A *database* is a set of relations. For example, $\{R(AB), R(BC), R(CD), R(DA)\}$ is a database. We use calligraphic letters, such as $\mathcal{D}$, to denote databases. Let $att(\mathcal{D})$ denote the set of all attributes in $\mathcal{D}$. For instance,

$$att(\{R(AB), R(BC), R(CD), R(DA)\}) = \{A, B, C, D\}.$$

A *database scheme* is a multiset of relation schemes, which may consist of several relation schemes having the same set of attributes. We use bold letters, such as **D**, to denote database schemes. The database scheme of a database $\{R(\mathbf{V}_1), \ldots, R(\mathbf{V}_n)\}$ is $\{\mathbf{V}_1, \ldots, \mathbf{V}_n\}$. For instance, the database scheme of $\{R(AB), R(BC), R(CD), R(DA)\}$ is $\{AB, BC, CD, DA\}$.

In order to explain some notions regarding database schemes, it is helpful to use a *hypergraph* in order to represent a database scheme, with the attributes as nodes, and the relation schemes as hyperedges. For instance, the hypergraph in Figure 1 represents $\{AB, BC, CD, DA\}$.

Two relation schemes, say $\mathbf{V}_1$ and $\mathbf{V}_k$, are *linked* if there is a sequence of relation schemes $\mathbf{V}_1, \ldots, \mathbf{V}_k$ such that $\mathbf{V}_i \cap \mathbf{V}_{i+1}$ is nonempty for $1 \leq i < k$. For instance, in Figure 1, AB and CD are linked, because AB, BC, CD meets the above condition. A database scheme is *connected* if any two relation schemes are linked. For example, $\{AB, BC, CD, DA\}$ is connected. A *component* **C** of a database scheme **D** is a subset of **D** such that any relation scheme in **C** and any relation scheme in $\mathbf{D} - \mathbf{C}$ are not linked. For instance, $\{AB, BC\}$ and $\{DE, EF\}$ are components of $\{AB, BC, DE, EF\}$.

A database is *connected* if its database scheme is connected. For instance,

$$\{R(AB), R(BC), R(CD), R(DA)\}$$

is connected, because its database scheme $\{AB, BC, CD, DA\}$ is connected. A *component* $\mathscr{C}$ of a database $\mathscr{D}$ is a subset of $\mathscr{D}$ such that the database scheme of $\mathscr{C}$ is a component of the database scheme of $\mathscr{D}$. For example, $\{R(AB), R(BC)\}$ and $\{R(DE), R(EF)\}$ are components of $\{R(AB), R(BC), R(DE), R(EF)\}$.

Let t be a tuple over a relation scheme $\mathbf{V}$ and let $\mathbf{X}$ be a subset of $\mathbf{V}$. $t[\mathbf{X}]$ denotes the restriction of the mapping t onto $\mathbf{X}$. For instance, if (1, 2, 3) is a tuple over ABC, (1, 2, 3) $[AB]$ means (1, 2). The *projection* of a relation $R(\mathbf{V})$ onto $\mathbf{X}$, denoted $\pi_{\mathbf{X}}(R(\mathbf{V}))$, is $\{t[\mathbf{X}]\ t \in R(\mathbf{V})\}$. When we write $\pi_{\mathbf{X}}R(\mathbf{V})$, $\mathbf{X}$ must be a subset of $\mathbf{V}$.

The (*natural*) *join* of relations $R(\mathbf{V})$ and $R(\mathbf{W})$, denoted $R(\mathbf{V}) \bowtie R(\mathbf{W})$, is

$$\{t \mid t \text{ is a tuple over } \mathbf{V} \cup \mathbf{W},\ t[\mathbf{V}] \in R(\mathbf{V}) \text{ and } t[\mathbf{W}] \in R(\mathbf{W})\}.$$

$\mu \in R(\mathbf{V})$ *agrees with* (or *joins with*) $\nu \in R(\mathbf{W})$ if $\mu[\mathbf{V} \cap \mathbf{W}] = \nu[\mathbf{V} \cap \mathbf{W}]$. The *semijoin* of $R(\mathbf{V})$ by $R(\mathbf{W})$, denoted $R(\mathbf{V}) \ltimes R(\mathbf{W})$, is $\pi_{\mathbf{V}}\ (R(\mathbf{V}) \bowtie R(\mathbf{W}))$.

2.2. JOIN EXPRESSIONS. A *join expression* is an expression with relations as operands, join ($\bowtie$) as a binary operator, and parentheses. A join expression over a database $\mathscr{D}$ is one in which each relation in $\mathscr{D}$ appears exactly once. For instance,

$$(R(AB) \bowtie R(BC)) \bowtie (R(CD) \bowtie R(DA))$$

is a join expression over $\{R(AB), R(BC), R(CD), R(DA)\}$. We can specify an ordering of joins by a join expression. For instance, the above join expression indicates that we execute $R(AB) \bowtie R(BC)$ and $R(CD) \bowtie R(DA)$ independently, and then perform the whole join.

Although join expressions over $\mathscr{D} = \{R(\mathbf{V}_1), \ldots, R(\mathbf{V}_k)\}$ take joins in different orders, owing to the commutativity and associativity of joins, all of them return the same result, and we will therefore denote this result $\bowtie \mathscr{D}$ (or $R(\mathbf{V}_1) \bowtie \cdots \bowtie R(\mathbf{V}_k)$).

Join expressions specify the order of the joins by means of parentheses; however, to clarify the order it is helpful to represent a join expression as a tree structure. The *join expression tree* T for a join expression E over $\mathscr{D}$ is a tree inductively defined as follows:

—Each node of T is a database.

—If E is a single relation $R(\mathbf{V})$, T has a single node $\{R(\mathbf{V})\}$.

—If E has the form of $E_1 \bowtie E_2$, the root of T is $\mathscr{D}$, and the root has two subtrees that are join expression trees for E_1 and E_2.

For instance, Figure 2 shows the join expression tree for

$$(R(AB) \bowtie R(BC)) \bowtie (R(CD) \bowtie R(DA)).$$

$E_1 \bowtie E_2$ is a *Cartesian product* if E_1 and E_2 do not share any attribute. For example, $R(AB) \bowtie (R(CE) \bowtie R(ED))$ is a Cartesian product, since no attribute appears both in $R(AB)$ and in $R(CE) \bowtie R(ED)$. A join expression is

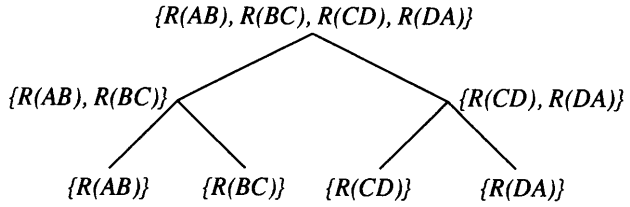


FIG. 2. Join expression tree for $(R(AB) \bowtie R(BC)) \bowtie (R(CD) \bowtie R(DA))$.

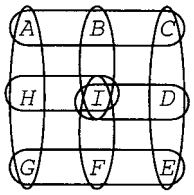


FIG. 3. Hypergraph for $\{ABC, CDE, EFG, GHA, BI, DI, FI, HI\}$.

Cartesian-product-free (or *CPF*, for short) if none of its joins is a Cartesian product. For instance, $(R(AB) \bowtie R(BC)) \bowtie (R(CD) \bowtie R(DA))$ is CPF. On the other hand, a join expression is non-CPF if it contains a Cartesian product. For instance, $(R(AB) \bowtie R(CD)) \bowtie (R(BC) \bowtie R(DA))$ is non-CPF. A join expression tree is *CPF* if its join expression is CPF.

As a measure of the cost of executing a join expression, we use the number of tuples that appear in the input relations or the relations generated during the execution. Let $|R(\mathbf{V})|$ denote the number of tuples in $R(\mathbf{V})$. We call $|R(\mathbf{V})|$ the *size* of $R(\mathbf{V})$. The *cost* of solving join expression E is defined as

$$\sum\{|F| \mid F \text{ is a subexpression of } E\}.$$

For instance, the cost of $(R(AB) \bowtie R(BC)) \bowtie (R(CD) \bowtie R(DA))$ is

$$\begin{aligned} &|(R(AB) \bowtie R(BC)) \bowtie (R(CD) \bowtie R(DA))| \\ &\quad + |R(AB) \bowtie R(BC)| + |R(CD) \bowtie R(DA)| \\ &\quad + |R(AB)| + |R(BC)| + |R(CD)| + |R(DA)|. \end{aligned}$$

The *cost* of solving the join expression tree T for E is defined as the cost of E. Put another way, the cost of T is

$$\sum\{|\bowtie \mathscr{X}| \mid \mathscr{X} \text{ is a node in } T\}.$$

There are many methods that take advantage of indexes, block sizes, and main memory sizes to compute joins. Although not ideal, our cost measure is reasonable, because when the cost is n the cost of the actual best possible method is no more than $O(n \log n)$ and no less than $\Omega(n)$ [Ullman 1989].

Example 2.2.1. Let $\mathscr{D}$ be the database

$$\{R(ABC), R(CDE), R(EFG), R(GHA), R(BI), R(DI), R(FI), R(HI)\}.$$

The hypergraph in Figure 3 shows the database scheme of $\mathscr{D}$.

$R(ABC)$	$R(CDE)$	$R(EFG)$	$R(GHA)$
$(a, 1, a)$	$(a, 1, a)$	$(a, 1, a)$	$(a, 1, b)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(a, 10^{3k}, a)$	$(a, 10^{2k}, a)$	$(a, 10^{k}, a)$	$(a, 10^{2k}, b)$
$(b, 1, b)$	$(b, 1, b)$	$(b, 1, b)$	$(b, 1, a)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(b, 10^{3k}, b)$	$(b, 10^{2k}, b)$	$(b, 10^{k}, b)$	$(b, 10^{2k}, a)$
(c, c, c)	(c, c, c)	(c, c, c)	(c, c, c)

FIG. 4. Database of $\{R(ABC), R(CDE), R(EFG), R(GHA)\}$.

We use the tuples in Figure 4 for $\{R(ABC), R(CDE), R(EFG), R(GHA)\}$. Let $R(m)$ denote $\{(x, y) \mid x = 1, \ldots, 10^{mk}, y = 1, \ldots, 10^{k}\} \cup \{(c, c)\}$, and define $R(BI) = R(3)$, $R(DI) = R(2)$, $R(FI) = R(1)$, and $R(HI) = R(2)$.

We assume that $k \geq 1$. Observe that $\mathcal{D}$ is *locally* (*pairwise*) *consistent*; that is, for any pair of relations $R(\mathbf{V})$ and $R(\mathbf{W})$, $\pi_{\mathbf{v}}(R(\mathbf{V}) \bowtie R(\mathbf{W}))$ is equal to $R(\mathbf{V})$. $\mathcal{D}$, however, is not *globally consistent*; that is, it is not true that for any relation $R(\mathbf{V})$, $\pi_{\mathbf{v}}(\bowtie \mathcal{D})$ is equal to $R(\mathbf{V})$. Actually $\bowtie \mathcal{D}$ has only one tuple such that the value of each attribute is c. This fact implies that it is useless to apply a semijoin program [Ullman 1989] to this database.

Among all join expressions, an optimal join expression is one that computes $((R(ABC) \bowtie R(EFG)) \bowtie (R(CDE) \bowtie R(GHA)))$ first and then takes the join of the running relation and each of $\{R(BI), R(DI), R(FI), R(HI)\}$ one by one. An example of such a join expression is:

$$(((((R(ABC) \bowtie R(EFG)) \bowtie (R(CDE) \bowtie R(GHA)))$$

$$\bowtie R(BI)) \bowtie R(DI)) \bowtie R(FI)) \bowtie R(HI),$$

and its cost is less than 10^{4k+1}. Note that the above join expression is nonlinear and non-CPF. It is left to the reader to show that the cost of any CPF (or any linear) join expression exceeds $2 \cdot 10^{5k}$.

Since we can increase k as much as we want, both the cheapest linear join expression and the cheapest CPF join expression are much worse than the optimal nonlinear and non-CPF join expression by orders of magnitudes as mentioned in Section 1. □

2.3. PROGRAMS. We now define programs that consist of joins, semijoins, and projections. A *relation scheme variable* $\mathbf{X}$ is a variable for storing a relation scheme. For instance, if $\mathbf{X}$ stores AB, $R(\mathbf{X})$ denotes a relation over AB. A statement has one of the following forms:

Project statement: $R(\mathbf{X}) := \pi_{Y} R(\mathbf{Z})$, where $\mathbf{Y} \subseteq \mathbf{Z}$.

Join statement: $R(\mathbf{X}) := R(\mathbf{Y}) \bowtie R(\mathbf{Z})$

Semijoin statement: $R(\mathbf{X}) := R(\mathbf{X}) \ltimes R(\mathbf{Y})$

In each statement, the left-hand side of the assignment operator (:=) is the *head*, and the right-hand side is the *body*. **X**, which appears in the head of each statement, must be a relation scheme variable. **Y** and **Z** could be either relation scheme variables or instances of relation schemes. Each statement destructively assigns the relation computed in the body to the head $R(\mathbf{X})$, and may also change the value of **X**. The project statement sets **X** to the value of **Y**, and the join statement sets **X** to the value of $\mathbf{Y} \cup \mathbf{Z}$, but the semijoin statement leaves **X** untouched. For instance, "$R(\mathbf{X}) := \pi_{AB}R(ABC)$" is a project statement. After the execution, $R(\mathbf{X}) = \pi_{AB}R(ABC)$, and $\mathbf{X} = AB$. Next, suppose that $R(\mathbf{Y}) = \pi_{AB}R(ABC)$. After the execution of "$R(\mathbf{Y}) := R(\mathbf{Y}) \bowtie R(BD)$,", $R(\mathbf{Y}) = (\pi_{AB}R(ABC)) \bowtie R(BD)$, and $\mathbf{Y} = ABD$.

A *program* over a database $\mathcal{D}$ is a sequence of statements, say $s_1, \ldots, s_n$, such that every $R(\mathbf{V})$ in the body of a statement (say s_k) must appear either in $\mathcal{D}$ or in the head of an earlier statement $s_j (j < k)$. This condition ensures that any relation scheme variable **V** is set to a relation scheme when we evaluate a statement having **V** in its body. For example, consider the following program:

$$R(\mathbf{X}) := R(AB) \bowtie R(BC), \qquad R(\mathbf{X}) := R(\mathbf{X}) \bowtie R(CD).$$

The first statement sets $R(\mathbf{X})$ to $R(AB) \bowtie R(BC)$, and the second sets $R(\mathbf{X})$ to $R(AB) \bowtie R(BC) \bowtie R(CD)$.

We define the cost of executing a program P over $\mathcal{D}$ in terms of the number of tuples in the input and intermediate relations. The *cost* of executing P is formally defined as

$$\sum\{|R| \mid R \text{ is an element of } \mathcal{D} \text{ or the head of a statement in } P\}.$$

The cost of "$R(\mathbf{X}) := R(AB) \bowtie R(BC)$, $R(\mathbf{X}) := R(\mathbf{X}) \bowtie R(CD)$", for instance, is

$$|R(AB)| + |R(BC)| + |R(CD)| + |R(AB) \bowtie R(BC)| + |R(AB) \bowtie R(BC) \bowtie R(CD)|.$$

We will use two complex statements sometimes for readability.

$$R(\mathbf{X}) := (\pi_{\mathbf{Y}}R(\mathbf{Z})) \ltimes R(\mathbf{U})$$

is a combination of the following projection statement and semijoin statement:

$$R(\mathbf{W}) := \pi_{\mathbf{Y}}R(\mathbf{Z}), \qquad R(\mathbf{X}) := R(\mathbf{W}) \ltimes R(\mathbf{U}),$$

where **W** is a new relation scheme variable that does not appear elsewhere.

$$R(\mathbf{X}) := R(\mathbf{X}) \bowtie \pi_{\mathbf{Y}}R(\mathbf{Z})$$

consists of the following projection statement and join statement:

$$R(\mathbf{W}) := \pi_{\mathbf{Y}}R(\mathbf{Z}), \qquad R(\mathbf{X}) := R(\mathbf{X}) \bowtie R(\mathbf{W}),$$

where **W** is a new relation scheme variable. When we calculate the cost of a program, complex statements are replaced with normal statements.

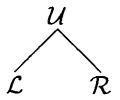


FIG. 5. Visit node $\mathcal{U}$ in T_1.

3. *Results*

We will present two algorithms in this section. Let $\mathcal{D}$ be a database such that $\bowtie \mathcal{D}$ is not empty. Without loss of generality, we can assume $\mathcal{D}$ is connected, because otherwise we would compute the join of each connected component of $\mathcal{D}$ individually and combine them. Given a join expression tree T_1 over $\mathcal{D}$, Algorithm 1 outputs a CPF join expression tree T_2 over $\mathcal{D}$. Given T_2, Algorithm 2 generates a program for computing $\bowtie \mathcal{D}$ such that the cost of the program is within a factor k of the cost of T_1, where k is polynomial in the number of relations in $\mathcal{D}$. Consider the case in which T_1 is optimal. We see that there exits a CPF join expression tree (namely T_2) from which Algorithm 2 derives a program whose cost is within a factor k of the cost of an optimal join expression tree. Neither Algorithm 1 nor Algorithm 2 use the actual tuples of $\mathcal{D}$, but they only consider the schemes of $\mathcal{D}$.

3.1. Deriving a CPF Join Expression Tree

Algorithm 1. Given a join expression tree T_1 over a connected database $\mathcal{D}$, this algorithm outputs a CPF join expression tree over $\mathcal{D}$.

Let us create a table that contains a CPF join expression tree over each component of every node in T_1. We visit each node of T_1 in some bottom-up order; that is, we visit each node after having visited all its children. Thus we begin at the leaves. Since each leaf is the CPF join expression tree over itself, we put all leaves into the table. We then visit an internal node, say $\mathcal{U}$, whose children have been visited. Let $\mathcal{L}$ and $\mathcal{R}$ be the left and right children of $\mathcal{U}$ respectively. See Figure 5.

Let $\mathcal{C}$ be an arbitrary component of $\mathcal{U}$. If the table already includes a tree rooted at $\mathcal{C}$, we add nothing to the table. Otherwise, $\mathcal{C}$ is neither a component of $\mathcal{L}$ nor a component of $\mathcal{R}$, and is therefore the union of some components of $\mathcal{L}$ and some components of $\mathcal{R}$. All components in $\mathcal{U}$ can be computed by the following steps:

(1) Let us regard components in $\mathcal{L}$ and components in $\mathcal{R}$ as nodes. Note that any two components in $\mathcal{L}(\mathcal{R})$ do not share any attribute. Generate a bipartite graph between $\mathcal{L}$ and $\mathcal{R}$ in which we draw an edge between a component in $\mathcal{L}$ and a component in $\mathcal{R}$ whenever the two components share an attribute. Observe that a connected component in the bipartite graph exactly corresponds to a component in $\mathcal{U}$.

(2) Visit all nodes in the bipartite graph in a depth-fist manner by scanning each edge only once, and output all connected component in the graph. Although there could be various ways of visiting nodes in a depth-first manner, we select one order arbitrarily, because we obtain the same set of nodes for the same component in $\mathcal{U}$.

Suppose that component $\mathcal{C}$ in $\mathcal{U}$ is found by visiting nodes for representing components $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m$ in this order. Note that any $\mathcal{C}_i (1 < i)$ has some common attributes with some earlier $\mathcal{C}_j (j < i)$. Since the table has a CPF join

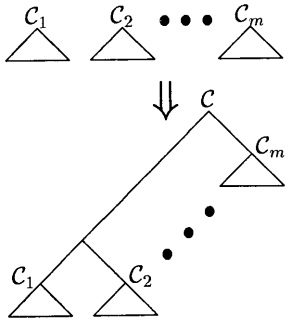


FIG. 6. Combining CPF join expression trees over $\mathcal{X}_i$ into one over $\mathcal{C}$.

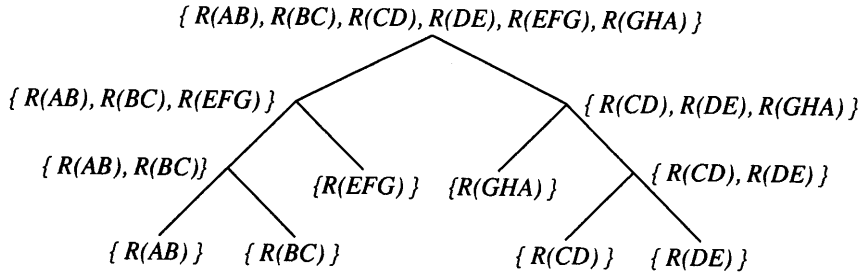


FIG. 7. Input to Algorithm 1.

expression tree over each $\mathcal{C}_i (1 \leq i \leq m)$, we combine those CPF join expression trees into a CPF join expression tree over $\mathcal{C}$, as shown in Figure 6.

Finally, after we have processed the root $\mathcal{D}$, since $\mathcal{D}$ is connected, we add to the table a CPF join expression tree over $\mathcal{D}$. □

Example 3.1.1. Let us apply Algorithm 1 to the tree in Figure 7, which has Cartesian products. First, we add to the table the databases at all leaves. Then we visit all internal nodes in bottom-up order; for instance, $\{R(AB), R(BC)\}$, $\{R(AB), R(BC), R(EFG)\}$, $\{R(CD), R(DE)\}$, $\{R(CD), R(DE), R(GHA)\}$, and the root.

—$\{R(AB), R(BC)\}$ has only one component, which is itself. Since the table does not contain any tree rooted at this component, we make a tree having $\{R(AB), R(BC)\}$ as its root, $\{R(AB)\}$ as its left child, and $\{R(BC)\}$ as its right child, and we add the tree to the table.

—$\{R(AB), R(BC), R(EFG)\}$ has two components, $\{R(AB), R(BC)\}$ and $\{R(EFG)\}$. The table already has trees rooted at them.

—$\{R(CD), R(DE)\}$ has only one component, which is itself. Since the table does not contain any tree rooted at this component, we make a tree having $\{R(CD), R(DE)\}$ as its root, $\{R(CD)\}$ as its left child, and $\{R(DE)\}$ as its right child, and we add the tree to the table. At this point, the table contains eight join expression trees, which are shown in Figure 8.

—$\{R(CD), R(DE), R(GHA)\}$ has two components, $\{R(CD), R(DE)\}$ and $\{R(GHA)\}$. The table already has trees rooted at them.

—The root has only one component, which is itself. As the table does not have a tree rooted at this component, we need to create such a tree. The left child of

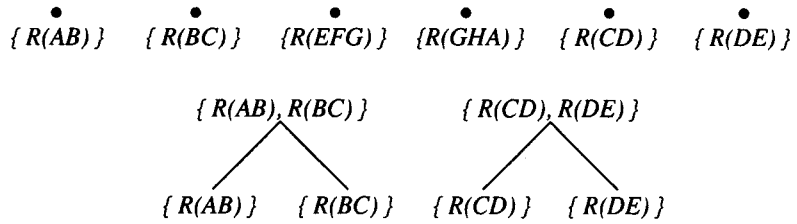


FIG. 8. Intermediate state of the table produced by Algorithm 1.

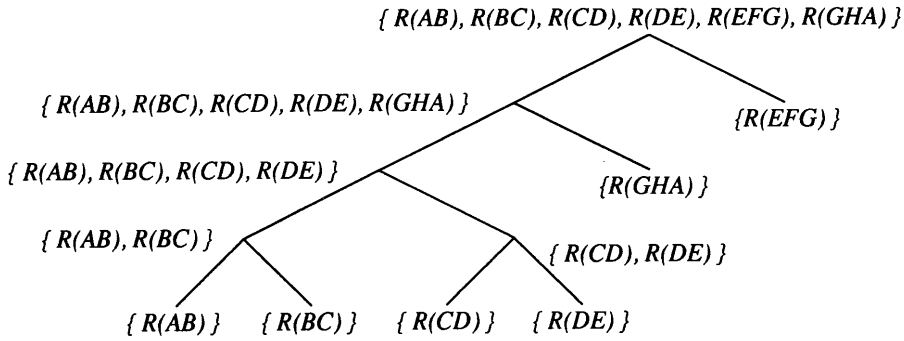


FIG. 9. Output from Algorithm 1.

$R(AB)$	$R(BC)$	$R(CD)$	$R(DE)$	$R(EFG)$	$R(GHA)$
$(a, 1)$	$(1, a)$	$(a, 1)$	$(1, a)$	$(a, 1, a)$	$(a, 1, b)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(a, 10^{3k})$	$(10^{3k}, a)$	$(a, 10^{2k})$	$(10^{2k}, a)$	$(a, 10^{k}, a)$	$(a, 10^{2k}, b)$
$(b, -1)$	$(-1, b)$	$(b, -1)$	$(-1, b)$	$(b, 1, b)$	$(b, 1, a)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(b, -10^{3k})$	$(-10^{3k}, b)$	$(b, -10^{2k})$	$(-10^{2k}, b)$	$(b, 10^{k}, a)$	$(b, 10^{2k}, a)$
(c, c)	(c, c)	(c, c)	(c, c)	(c, c, c)	(c, c, c)

FIG. 10. Database of Example 3.1.1.

the root has two components, $\{R(AB), R(BC)\}$ and $\{R(EFG)\}$, and the right child has two components $\{R(CD), R(DE)\}$ and $\{R(GHA)\}$. Since the table has trees rooted at those four components (see Figure 8), we combine those four trees. There are 16 alternative ways of combining those trees, and we can select an arbitrary one. Figure 9 shows one instance. Observe that we cannot create a linear join expression tree from those four trees.

The cost of the tree in Figure 9 could be significantly greater than the cost of the tree in Figure 7 in some databases. For instance, consider the tuples shown in Figure 10. □

Here we discuss some properties of Algorithm 1.

The reader might imagine that if we make right choices in Algorithm 1, we can always generate a linear and CPF join expression tree, but Example 3.1.1 gives a counter-example.

Example 3.1.1 also shows that the cost of a join expression tree T_2 produced by Algorithm 1 could be much greater than the cost of the original tree T_1. Algorithm 2, which will be presented in Subsection 3.2, derives from T_2 a program whose cost is within a factor k of the cost of T_1, where k is polynomial in the number of relations.

The computation time of Algorithm 1 is polynomial in the number of relations and the number of attributes. When we visit $\mathcal{U}$, suppose that the number of relations in $\mathcal{U}$ ($\mathcal{L}$, $\mathcal{R}$, respectively) is $r_{\mathcal{U}}(r_{\mathcal{L}}, r_{\mathcal{R}})$. Step (1) tests if each pair of a component in $\mathcal{L}$ and a component in $\mathcal{R}$ shares an attribute. Let a denote the number of attributes, and then this test requires $O(a \ln a)$ time by sorting attributes in each component. This test is performed for $r_{\mathcal{L}}r_{\mathcal{R}}$ pairs. Thus, Step (1) creates at most $r_{\mathcal{L}}r_{\mathcal{R}}$ edges, each edge of which is scanned only once by Step (2). Let $f(r_{\mathcal{L}})$ ($f(r_{\mathcal{R}})$, $f(r_{\mathcal{U}})$, respectively) denote the number of all pairs of components tested during Algorithm 1 visits all nodes in the subtree rooted at $\mathcal{L}$ ($\mathcal{R}$, $\mathcal{U}$). Since $r_{\mathcal{U}} = r_{\mathcal{L}} + r_{\mathcal{R}}$, we have

$$f(r_{\mathcal{U}}) = f(r_{\mathcal{L}} + r_{\mathcal{R}}) \leq r_{\mathcal{L}}r_{\mathcal{R}} + f(r_{\mathcal{L}}) + f(r_{\mathcal{R}}).$$

Note that $f(1) = 0$ and $f(2) = 1$. We can prove $f(i) < i^2/2$ by the induction on i.

Let us consider the case in which Algorithm 1 takes a CPF tree as input. As each database in a CPF join expression tree is connected, when we visit a node $\mathcal{V}$, the table has two trees rooted at $\mathcal{V}$'s two children. Thus, there are two alternative ways of combining those two trees to make a tree rooted at $\mathcal{V}$. Consequently, Algorithm 1 may output a tree that is obtained from the input CPF tree by swapping subtrees at some nodes. But observe that the cost of the input tree is equal to that of the output tree for any database.

3.2. Deriving a Program from a CPF Tree. Before presenting Algorithm 2 we will introduce projections of a special kind, called *core relations*, which are frequently used in Algorithm 2.

Relation scheme $\mathbf{V}$ is a *core scheme* of a set of relation schemes $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$ if for any distinct pair of $\mathbf{W}_i$ and $\mathbf{W}_j$, $\mathbf{W}_i \cap \mathbf{W}_j \subseteq \mathbf{V}$, and $\mathbf{V} \subseteq \mathbf{W}_0 \cup \cdots \cup \mathbf{W}_n$. The intersection of all core schemes of $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$ is also a core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$, and it is called the *least core scheme* of $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$. We can easily see that the least core scheme is the set of attributes each of which appears in at least two distinct relation schemes in $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$.

$\pi_{\mathbf{V}}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n))$ is a *core relation* of $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_n)\}$ if $\mathbf{V}$ is a core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$. The projection is the *least core relation* if $\mathbf{V}$ is the least core scheme.

For instance, $BCDE$ is a core scheme of $\{AB, BC, CD, CE\}$, and BC is the least core scheme. $\pi_{BCDE}(R(AB) \bowtie R(BC) \bowtie R(CD) \bowtie R(CE))$ is a core relation of $\{R(AB), R(BC), R(CD), R(CE)\}$, and $\pi_{BC}(R(AB) \bowtie R(BC) \bowtie R(CD) \bowtie R(CE))$ is the least core relation.

Algorithm 2. Let $\mathcal{D}$ be a connected database such that $\bowtie \mathcal{D}$ is not empty, and let T_1 be a join expression tree over $\mathcal{D}$. Suppose that Algorithm 1 takes T_1 as input and outputs a CPF join expression tree T_2. From T_2, we will create a program for computing $\bowtie \mathcal{D}$.

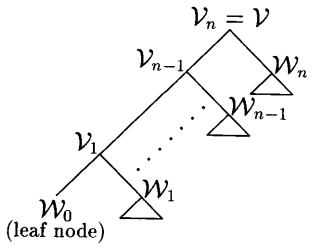


FIG. 11. Visiting node $\mathcal{V}$ in T_2.

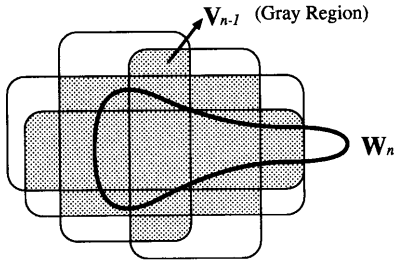


FIG. 12. Hypergraph for Case 2(a).

Visit all internal nodes of T_2 in bottom-up order; that is, visit each node after visiting all its children. Suppose that we visit the node $\mathcal{V}$ shown in Figure 11. We will create a sequence of statements for computing the least core relation of $\{\bowtie \mathcal{W}_0, \ldots, \bowtie \mathcal{W}_n\}$. Furthermore, if $\mathcal{V}$ is the right child of its parent, we will also generate a sequence of statements for computing $\bowtie \mathcal{V}$.

We will assume that for each $i = 0, \ldots, n$, $\bowtie \mathcal{W}_i$ is stored in $R(\mathbf{W}_i)$. If $\mathcal{W}_i$ is a leaf node, assume that $\mathcal{W}_i = \{R(\mathbf{W}_i)\}$, which implies that $\bowtie \mathcal{W}_i = R(\mathbf{W}_i)$. If $\mathcal{W}_i$ is an internal node, $\mathcal{W}_i$ is the right child of its parent, and hence $\mathcal{W}_i$ has been visited. Thus, we have created a sequence of statements for computing $\bowtie \mathcal{W}_i$, and assume that $\bowtie \mathcal{W}_i$ is stored in $R(\mathbf{W}_i)$.

We now create a sequence of statements for computing the least core relation of $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_n)\}$ that is set to $R(\mathbf{V}_n)$.

(1) When $n = 1$, create

$$R(\mathbf{V}_1) := (\pi_{\mathbf{W}_0 \cap \mathbf{W}_1} R(\mathbf{W}_0)) \ltimes R(\mathbf{W}_1).$$

From Proposition A.1 in Appendix A, $R(\mathbf{V}_1)$ is set to the least core relation of $\{R(\mathbf{W}_0), R(\mathbf{W}_1)\}$.

(2) When $n > 1$, since we have visited $\mathcal{V}_{n-1}$, assume that $R(\mathbf{V}_{n-1})$ stores the least core relation of $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_{n-1})\}$. We first test whether $\mathbf{V}_{n-1}$ is the least core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_{n-1}, \mathbf{W}_n\}$. Since $\mathbf{V}_{n-1}$ is assumed to be the least core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_{n-1}\}$, we only need to check whether $\mathbf{W}_i \cap \mathbf{W}_n \subseteq \mathbf{V}_{n-1}$ for each $i = 0, \ldots, n-1$,

(a) If $\mathbf{V}_{n-1}$ is the core scheme, create

$$R(\mathbf{V}_n) := R(\mathbf{V}_{n-1}) \ltimes R(\mathbf{W}_n)$$

Figure 12 illustrates the hypergraph for $\{\mathbf{W}_0, \ldots, \mathbf{W}_{n-1}, \mathbf{W}_n\}$. The gray region shows $\mathbf{V}_{n-1}$, and the region enclosed in the bold line represents

$\mathbf{W}_n$. From Proposition A.1 in Appendix A, this statement sets $R(\mathbf{V}_n)$ to the least core relation of $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_n)\}$.

(b) Otherwise, let $\mathcal{F}$ be $\{R(\mathbf{W}_j) \mid 0 \leq j < n, \mathbf{W}_j \cap \mathbf{W}_n \neq \phi\}$, and generate the following statements:

$$R(\mathbf{X}) := \pi_{\mathbf{V}_{n-1} \cap att(\mathcal{F})} R(\mathbf{V}_{n-1})$$

$$R(\mathbf{X}) := R(\mathbf{X}) \bowtie (\pi_{\mathbf{W}_j \cap (\mathbf{V}_{n-1} \cup \mathbf{W}_n)} R(\mathbf{W}_j)),$$

for each $R(\mathbf{W}_j) \in \mathcal{F}$ such that $(\mathbf{W}_j \cap \mathbf{W}_n) - \mathbf{V}_{n-1} \neq \phi$

$$R(\mathbf{X}) := R(\mathbf{X}) \ltimes R(\mathbf{W}_n)$$

$$R(\mathbf{V}_n) := R(\mathbf{V}_{n-1}) \bowtie R(\mathbf{X})$$

From Proposition A.2 in Appendix A, the last statement sets $R(\mathbf{V}_n)$ to the least core relation of $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_n)\}$. For instance, let us consider the case when $\mathcal{F} = \{\mathbf{W}_{j1}, \mathbf{W}_{j2}\}$, which is illustrated in Figure 13. The gray region in each picture shows the value of $\mathbf{X}$ or $\mathbf{V}_n$ after the execution of each of the above statements.

(3) Furthermore, if $\mathcal{V}$ is the right child of $\mathcal{V}$'s parent or the root of T_2, for each $\mathbf{W}_i$ that is not a subset of $\mathbf{V}_n$ create

$$R(\mathbf{V}_n) := R(\mathbf{V}_n) \bowtie R(\mathbf{W}_i).$$

From Proposition A.3 in Appendix, the last statement sets $R(\mathbf{V}_n)$ to $R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)$, which is equal to $\bowtie \mathcal{V}$.

Repeat the above steps until we visit the root $\mathcal{D}$ of T_2; we then have a program for computing $\bowtie \mathcal{D}$. □

Example 3.2.1. Let us apply Algorithm 2 to the CPF join expression tree in Figure 9, which Algorithm 1 derives from the non-CPF join expression tree in Figure 7.

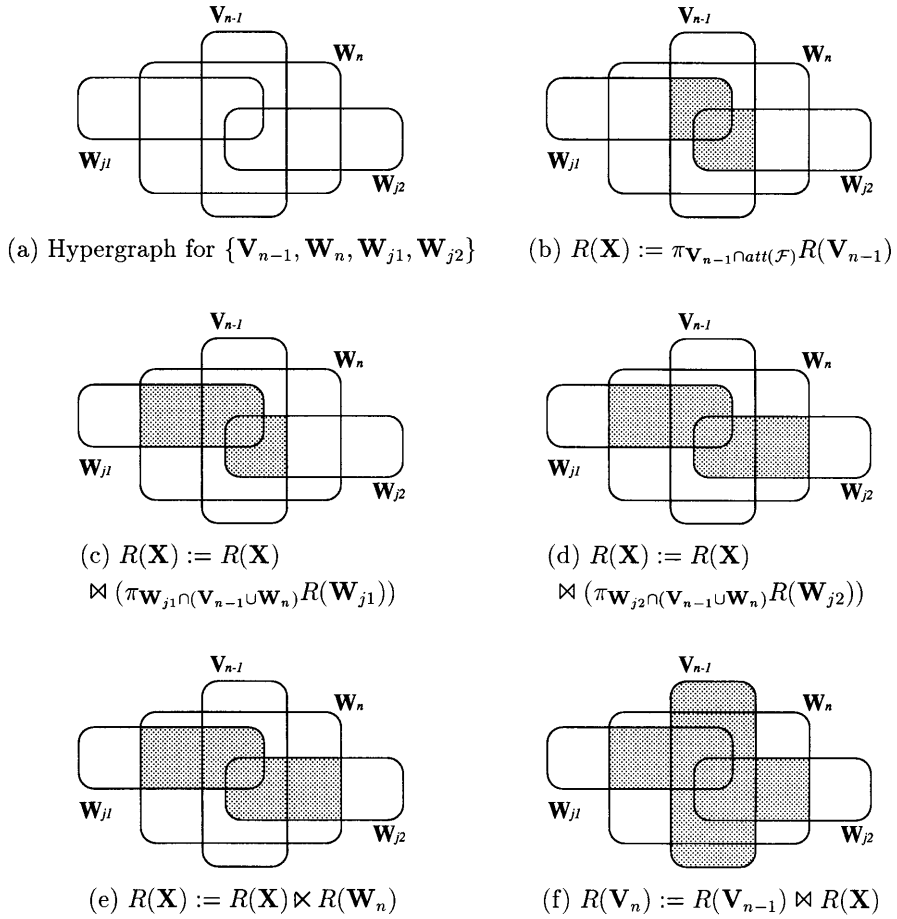
Consider the case in which we visit the root in Figure 9 after having visited all the other internal nodes. Since we have visited $\{R(CD), R(DE)\}$, which is the right child of its parent, suppose that

$$R(CDE) = R(CD) \bowtie R(DE).$$

Since we also have visited the left child of the root, assume that $R(ABC)$ stores the least core relation of $\{R(AB), R(BC), R(CDE), R(GHA)\}$; that is,

$$R(ABC) = \pi_{ABC}(R(AB) \bowtie R(BC) \bowtie R(CDE) \bowtie R(GHA)).$$

Now we will create a sequence of statements for computing the least core relation of $\{R(AB), R(BC), R(CDE), R(GHA), R(EFG)\}$. Since ABC is not the least core relation of $\{AC, BC, CDE, GHA, EFG\}$ (the hypergraph of which is shown in Figure 14), we apply Case 2(b). In this case, $\mathcal{F}$ is $\{R(CDE),$



(a) Hypergraph for $\{\mathbf{V}_{n-1}, \mathbf{W}_n, \mathbf{W}_{j1}, \mathbf{W}_{j2}\}$

(b) $R(\mathbf{X}) := \pi_{\mathbf{V}_{n-1} \cap att(\mathcal{F})} R(\mathbf{V}_{n-1})$

(c) $R(\mathbf{X}) := R(\mathbf{X}) \ltimes (\pi_{\mathbf{W}_{j1} \cap (\mathbf{V}_{n-1} \cup \mathbf{W}_n)} R(\mathbf{W}_{j1}))$

(d) $R(\mathbf{X}) := R(\mathbf{X}) \ltimes (\pi_{\mathbf{W}_{j2} \cap (\mathbf{V}_{n-1} \cup \mathbf{W}_n)} R(\mathbf{W}_{j2}))$

(e) $R(\mathbf{X}) := R(\mathbf{X}) \ltimes R(\mathbf{W}_n)$

(f) $R(\mathbf{V}_n) := R(\mathbf{V}_{n-1}) \ltimes R(\mathbf{X})$

FIG. 13. Example of Case 2(b).

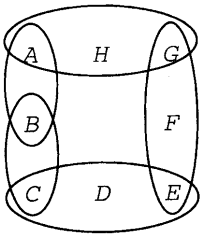


FIG. 14. Hypergraph for $\{AB, BC, CDE, EFG, GHA\}$.

$R(GHA)\}$, because each relation has a common attribute with EFG. Thus, we generate

$$
\begin{aligned}
R(\mathbf{X}) &:= \pi_{ABC \cap (CDE \cup GHA)} R(ABC) \\
&\quad (= \pi_{AC}(R(AB) \bowtie R(BC) \bowtie R(CDE) \bowtie R(GHA)) \\
&\quad \subseteq \pi_{AC}(R(CD) \bowtie R(DE) \bowtie R(GHA))) \\
R(\mathbf{X}) &:= R(\mathbf{X}) \bowtie \pi_{CDE \cap (ABC \cup EFG)} R(CDE) \\
&\quad (= \pi_{ACE}(R(AB) \bowtie R(BC) \bowtie R(CDE) \bowtie R(GHA)) \\
&\quad \subseteq \pi_{ACE}(R(CD) \bowtie R(DE) \bowtie R(GHA))) \\
R(\mathbf{X}) &:= R(\mathbf{X}) \bowtie \pi_{GHA \cap (ABC \cup EFG)} \mathrm{R(GHA)} \\
&\quad (= \pi_{ACEG}(R(AB) \bowtie R(BC) \bowtie R(CDE) \bowtie R(GHA)) \\
&\quad \subseteq \pi_{ACEG}(R(CD) \bowtie R(DE) \bowtie R(GHA))) \\
R(\mathbf{X}) &:= R(\mathbf{X}) \ltimes R(EFG) \\
&\quad (= \pi_{ACEG}(R(AB) \bowtie R(BC) \bowtie R(CDE) \bowtie R(GHA) \bowtie R(EFG)) \\
&\quad \subseteq \pi_{ACEG}(R(CD) \bowtie R(DE) \bowtie R(GHA))) \\
R(\mathbf{V}_4) &:= R(ABC) \bowtie R(\mathbf{X}) \\
&\quad (= \pi_{ABCEG}(R(AB) \bowtie R(BC) \bowtie R(CDE) \bowtie R(GHA) \bowtie R(EFG)) \\
&\quad \subseteq \pi_{ABCEG}(R(AB) \bowtie R(BC) \bowtie R(EFG)))
\end{aligned}
$$

In the above, we show the relation of $R(\mathbf{X})$ or $R(\mathbf{V}_4)$ after the execution of each statement. We will explain the details of how to compute those relations in the proof of Proposition A.2 in Appendix A. We also present a superset of each relation. Since $\{R(CD), R(DE), R(GHA)\}$ and $\{R(AB), R(BC), R(EFG)\}$ are nodes in Figure 7, throughout the execution of the above statements, $|R(\mathbf{X})|$ and $|R(\mathbf{V}_4)|$ are always bounded by the size of the join of a node in Figure 7.

Finally, since neither CDE, EFG, nor GHA is a subset of $\mathbf{V}_4 (= ABCEG)$, we apply Case (3) of Algorithm 2. Let X denote

$$R(AB) \bowtie R(BC) \bowtie R(CD) \bowtie R(DE) \bowtie R(EFG) \bowtie R(GHA).$$

$R(\mathbf{V}_4)$ now stores $\pi_{ABCEG} X$, and we create the following statements:

$$
\begin{aligned}
R(\mathbf{V}_4) &:= R(\mathbf{V}_4) \bowtie R(CDE) (= \pi_{ABCDEG} X) \\
R(\mathbf{V}_4) &:= R(\mathbf{V}_4) \bowtie R(EFG) (= \pi_{ABCDEFG} X) \\
R(\mathbf{V}_4) &:= R(\mathbf{V}_4) \bowtie R(GHA) (= X).
\end{aligned}
$$

Throughout the execution of the above statements, $|R(\mathbf{V}_4)|$ is bounded by $|X|$, where X is the join of the root in Figure 7. □

In the above example, the size of the head of each statement can be successfully bounded by the size of the join of a node in Figure 7. We will generalize this property.

LEMMA 3.2.2. *Let $\mathcal{D}$ be a connected database such that $\bowtie \mathcal{D} \neq \phi$, and let T_1 be a join expression tree over $\mathcal{D}$. Let T_2 be a CPF join expression tree that is produced by Algorithm* 1 *from T_1. Suppose that Algorithm* 2 *takes T_2 as input and outputs a program for computing $\bowtie \mathcal{D}$. For each statement in the program, T_1 has a node $\mathcal{X}$ such that the size of the head of the statement is bounded by $\bowtie \mathcal{X}$.*

PROOF. The proof is an induction on the number of internal nodes visited. Suppose that we visit $\mathcal{V}$ in Figure 11. Assume that $R(\mathbf{W}_i)$ stores $\bowtie \mathcal{W}_i$ for each $i = 0, \ldots, n$.

When $n = 1$, Algorithm 2 generates the following statement:

$$R(\mathbf{V}_1) := (\pi_{\mathbf{W}_0 \cap \mathbf{W}_1} R(\mathbf{W}_0)) \ltimes R(\mathbf{W}_1).$$

Since this is a semijoin statement, $|R(\mathbf{V}_1)| \leq |\pi_{\mathbf{W}_0 \cap \mathbf{W}_1} R(\mathbf{W}_0)| \leq |R(\mathbf{W}_0)|$. We have assumed that $R(\mathbf{W}_0) = \bowtie \mathcal{W}_0$, and hence $|R(\mathbf{V}_1)| \leq |\bowtie \mathcal{W}_0|$. Since $\mathcal{W}_0$ is also a leaf node in T_1, our claim is proved for this case.

Consider the case in which $n > 1$. If $\mathbf{V}_{n-1}$ is the least core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$, Algorithm 2 generates

$$R(\mathbf{V}_n) := R(\mathbf{V}_{n-1}) \ltimes R(\mathbf{W}_n).$$

Since this is a semijoin statement, $|R(\mathbf{V}_n)| \leq |R(\mathbf{V}_{n-1})|$. From the inductive hypothesis, T_1 has a node $\mathcal{X}$ such that $|R(\mathbf{V}_{n-1})| \leq |\bowtie \mathcal{X}|$, and hence $|R(\mathbf{V}_n)| \leq |\bowtie \mathcal{X}|$.

Next suppose that $\mathbf{V}_{n-1}$ is not the least core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$. Let $\mathcal{F}$ be $\{R(\mathbf{W}_j) \mid 0 \leq j < n, \mathbf{W}_j \cap \mathbf{W}_n \neq \phi\}$. Algorithm 2 generates

$$\begin{aligned}
R(\mathbf{X}) &:= \pi_{\mathbf{V}_{n-1} \cap att(\mathcal{F})} R(\mathbf{V}_{n-1}) \\
R(\mathbf{X}) &:= R(\mathbf{X}) \bowtie (\pi_{\mathbf{W}_j \cap (\mathbf{V}_{n-1} \cup \mathbf{W}_n)} R(\mathbf{W}_j)), \\
&\qquad \text{for each } R(\mathbf{W}_j) \in \mathcal{F} \text{ such that } (\mathbf{W}_j \cap \mathbf{W}_n) - \mathbf{V}_{n-1} \neq \phi \\
R(\mathbf{X}) &:= R(\mathbf{X}) \ltimes R(\mathbf{W}_n) \\
R(\mathbf{V}_n) &:= R(\mathbf{V}_{n-1}) \bowtie R(\mathbf{X})
\end{aligned}$$

From Proposition A.2 in Appendix A, throughout the execution of the above statements

$$|R(\mathbf{X})| \leq |\pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})} (R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}))|,$$

and $R(\mathbf{V}_n)$ is set to the least core relation of $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_{n-1}), R(\mathbf{W}_n)\}$. Since $\mathbf{V}_{n-1}$ is the least core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_{n-1}, \mathbf{W}_n)\}$, $\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathcal{F}))$ is the least core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_{n-1}, \mathbf{W}_n\}$, and therefore

$$R(\mathbf{V}_n) = \pi_{\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathcal{F}))} (R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}) \bowtie R(\mathbf{W}_n)).$$

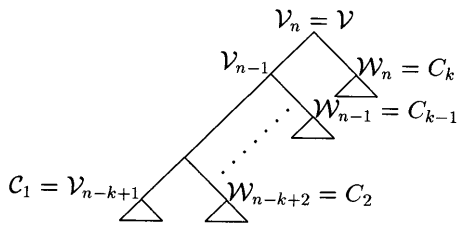


FIG. 15. How $\mathcal{V}$ is created by Algorithm 1.

Suppose that $\mathcal{V}$ is generated by Algorithm 1 when a node $\mathcal{U}$ in T_1 is visited. Let $\mathcal{L}$ and $\mathcal{R}$ be the left and right children of $\mathcal{U}$, respectively. There exists a sequence of some components of $\mathcal{L}$ or $\mathcal{R}$, say $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_k$, such that

$$\mathcal{C}_1 = \mathcal{V}_{n-k+1}(or\ \mathcal{W}_0),\ \mathcal{C}_2 = \mathcal{W}_{n-k+2}, \ldots, \mathcal{C}_k = \mathcal{W}_n.$$

See Figure 15.

We prove that $|R(\mathbf{X})|$ is bounded by $|\bowtie \mathcal{L}|$ or $|\bowtie \mathcal{R}|$. Observe that for any $R(\mathbf{W}_j)$ in $\mathcal{F}$, there exists $\mathcal{C}_x (1 \leq x \leq k-1)$ such that $\mathbf{W}_j \subseteq att(\mathcal{C}_x)$. As $\mathbf{W}_j \cap \mathbf{W}_n \neq \phi$, $\mathcal{C}_x$ and $\mathcal{C}_k$ share common attributes. If $\mathcal{C}_k$ is a component of $\mathcal{L}(\mathcal{R})$, $\mathcal{C}_x$ is a component of $\mathcal{R}(\mathcal{L})$, because no distinct components of a database share any common attributes. Thus, $att(\mathcal{F})$ is a subset of $att(\mathcal{L}) \cap att(\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_{k-1})$ or is a subset of $att(\mathcal{R}) \cap att\ (\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_{k-1})$. In the former case, we can prove that $|R(\mathbf{X})| \leq |\bowtie \mathcal{L}|$ as follows:

$$|R(\mathbf{X})|$$

$$\leq |\pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})}(R(\mathbf{W}_0) \bowtie \cdots \bowtie \mathrm{R}(\mathbf{W}_{n-1}))|$$

From Proposition A.2 in Appendix A.

$$= |\pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})} \bowtie (\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_{k-1})|$$

Since $R(\mathbf{W}_i) = \bowtie \mathcal{W}_i$ and $\mathcal{W}_0 \cup \cdots \cup \mathcal{W}_{n-1} = \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_{k-1}$.

$$\leq |\pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})} \bowtie (\cup \{\mathcal{C}_i | 1 \leq i \leq k-1, \mathcal{C}_i \text{ is a component of } \mathcal{L}\})|$$

Since $(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F}) \subseteq att(\mathcal{F}) \subseteq att(\mathcal{L}) \cap att(\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_{k-1})$.

$$\leq |\bowtie (\cup \{\mathcal{C}_i | 1 \leq i \leq k-1, \mathcal{C}_i \text{ is a component of } \mathcal{L}\})|$$

$$\leq |\bowtie \mathcal{L}|$$

The last statement holds, because $|\bowtie \mathcal{L}| = \Pi_{\mathcal{C} \text{ is a component of } \mathcal{L}} |\bowtie \mathcal{C}|$, and $|\bowtie \mathcal{C}| \geq 1$ from the assumption that $\bowtie \mathcal{D}$ is not empty. When $att(\mathcal{F})$ is a subset of $att(\mathcal{R}) \cap att(\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_{k-1})$, similarly we can prove that $|R(\mathbf{X})| \leq |\bowtie \mathcal{R}|$.

Next we prove that $|R(\mathbf{V}_n)| \leq |\bowtie \mathcal{L}|$ or $|R(\mathbf{V}_n)| \leq |\bowtie \mathcal{R}|$. Since $\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathcal{F}))$ is the least core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$, any attribute X in $\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathcal{F}))$ appears in at least two distinct relation schemes of $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$. As $R(\mathbf{W}_i) = \bowtie \mathcal{W}_i$, $\mathbf{W}_i$ is equal to $att(\mathcal{W}_i)$, and hence we may observe that

—X appears in $\mathcal{C}_1 (= \mathcal{W}_0 \cup \cdots \cup \mathcal{W}_{n-k+1})$, or

—X does not appear in $\mathscr{C}_1$, but appears in $\mathscr{C}_x$ and in $\mathscr{C}_y$, where $1 < x < y \leq k$.

In the latter case, either $\mathscr{C}_x$ or $\mathscr{C}_y$ is a component of $\mathscr{L}$, and the other is a component of $\mathscr{R}$, because no two distinct components of a database share any common attributes, and hence X appears in both $\mathscr{L}$ and $\mathscr{R}$. Thus, if $\mathscr{C}_1$ is a component of $\mathscr{L}$, X must appear in $\mathscr{L}$, and therefore

$$\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathscr{F})) \subseteq att(\mathscr{L}) \cap att(\mathscr{C}_1 \cup \cdots \cup \mathscr{C}_k).$$

If $\mathscr{C}_1$ is a component of $\mathscr{R}$, we have

$$\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathscr{F})) \subseteq att(\mathscr{R}) \cap att(\mathscr{C}_1 \cup \cdots \cup \mathscr{C}_k).$$

In the former case, we can prove that $|R(\mathbf{V}_n)| \leq |\bowtie \mathscr{L}|$ as follows:

$$|R(\mathbf{V}_n)|$$

$$= |\pi_{\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathscr{F}))}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n))|$$

From Proposition A.2 in Appendix A.

$$= |\pi_{\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathscr{F}))} \bowtie (\mathscr{C}_1 \cup \cdots \cup \mathscr{C}_k)|$$

Since $R(\mathbf{W}_i) = \bowtie \mathscr{W}_i$ and $\mathscr{W}_0 \cup \cdots \cup \mathscr{W}_n = \mathscr{C}_1 \cup \cdots \cup \mathscr{C}_k$

$$\leq |\pi_{\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathscr{F}))} \bowtie (\cup \{\mathscr{C}_i | 1 \leq i \leq k, \mathscr{C}_i \text{ is a component of } \mathscr{L}\})|$$

Since $\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathscr{F})) \subseteq att(\mathscr{L}) \cap att(\mathscr{C}_1 \cup \cdots \cup \mathscr{C}_k)$

$$\leq |\bowtie (\cup \{\mathscr{C}_i | 1 \leq i \leq k, \mathscr{C}_i \text{ is a component of } \mathscr{L}\})|$$

$$\leq |\bowtie \mathscr{L}|$$

The last statement holds, because $|\bowtie \mathscr{L}| = \Pi_{\mathscr{C} \text{ is a component of } \mathscr{L}} |\bowtie \mathscr{C}|$, and $|\bowtie \mathscr{C}| \geq 1$ from the assumption that $\bowtie \mathscr{D}$ is not empty. Similarly, we can prove that $|R(\mathbf{V}_n)| \leq |\bowtie \mathscr{R}|$, when $\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathscr{F})) \subseteq att(\mathscr{R}) \cap att(\mathscr{C}_1 \cup \cdots \cup \mathscr{C}_k)$.

Finally consider the case in which $\mathscr{V}$ is the right child of $\mathscr{V}$'s parent. For each $\mathbf{W}_i$ that is not a subset of $\mathbf{V}_n$, Algorithm 2 creates

$$R(\mathbf{V}_n) := R(\mathbf{V}_n) \bowtie R(\mathbf{W}_i).$$

From Proposition A.3 in Appendix A, throughout the execution of the above statements,

$$|R(\mathbf{V}_n)| \leq |R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)|.$$

Since $R(\mathbf{W}_i) = \bowtie \mathscr{W}_i$ and $\mathscr{V} = \mathscr{V}_n = \mathscr{W}_0 \cup \cdots \cup \mathscr{W}_n$,

$$R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n) = \bowtie \mathscr{V}.$$

Since $\mathscr{V}$ is the right child of $\mathscr{V}$'s parent, T_1 has a node, say $\mathscr{U}$, such that $\mathscr{V}$ is a component of $\mathscr{U}$, and therefore $|\bowtie \mathscr{V}| \leq |\bowtie \mathscr{U}|$ from the assumption that $\bowtie \mathscr{D}$ is not empty. Consequently, we have $|R(\mathbf{V}_n)| \leq |\bowtie \mathscr{U}|$. □

THEOREM 3.2.3. *Let $\mathcal{D}$ be a connected database such that $\bowtie \mathcal{D} \neq \phi$, and let T_1 be a join expression tree over $\mathcal{D}$. Let T_2 be a CPF join expression tree that is produced by Algorithm* 1 *from T_1. Suppose that Algorithm* 2 *takes T_2 as input and outputs a program P for computing $\bowtie \mathcal{D}$. Let r be the number of relations in $\mathcal{D}$. The cost of P is within a factor $r^2 + 3r - 6$ of the cost of T_1.*

PROOF. From Lemma 3.2.2, the size of the head of each statement in P is bounded by the size of the join of a node in T_1, and is also bounded by the cost of T_1, because the cost of T_1 is the sum of the size of the join of every node in T_1.

It remains for us to prove that the number of statements in P is less than or equal to $r^2 + 3r - 6$. Let $\mathcal{V}$ be a node in T_2 that is either the root or the right child of its parent. Let n be the number of internal nodes on the leftmost branch from $\mathcal{V}$ to the leaf node. Let $\mathcal{V}_1, \ldots, \mathcal{V}_n$ be the sequence of the internal nodes on the leftmost branch. See Figure 11.

When we visit $\mathcal{V}_1$, we generate two statements, because

$$R(\mathbf{V}_1) := (\pi_{\mathbf{W}_0 \cap \mathbf{W}_1} R(\mathbf{W}_0)) \ltimes R(\mathbf{W}_1)$$

consists of a projection statement and a semijoin statement. When we visit $\mathcal{V}_i (1 < i)$, we create at most $2i + 3$ statements, because

$$R(\mathbf{X}) := R(\mathbf{X}) \bowtie (\pi_{\mathbf{W}_j \cap (\mathbf{V}_{n-1} \cup \mathbf{W}_n)} R(\mathbf{W}_j))$$

also include a projection statement and a join statement. When we visit $\mathcal{V}_n (= \mathcal{V})$, additionally we create at most $n + 1$ statements. Thus, in total, we create at most $n^2 + 5n - 2$ statements, because

$$2 + \sum_{i=2}^{n} (2i + 3) + (n + 1) = n^2 + 5n - 2.$$

Let f be a function defined as $f(n) = n^2 + 5n - 2$.

Suppose that T_2 has k internal nodes each of which is either the root or the right child of its parent. Number those nodes from 1 to k, and let n_j be the number of internal nodes on the leftmost branch from the jth node. Then the number of statements in P is at most $f(n_1) + \cdots + f(n_k)$. Observe that $r - 1 = n_1 + \cdots + n_k$, and $f(x) + f(y) < f(x + y)$ for any positive integers x and y. Thus, we have

$$f(n_1) + \cdots + f(n_k) \leq f(n_1 + \cdots + n_k) = f(r - 1) = r^2 + 3r - 6. \qquad \square$$

The following corollary is a direct consequence of Theorem 3.2.3.

COROLLARY 3.2.4. *Let $\mathcal{D}$ be a connected database such that $\bowtie \mathcal{D}$ is not empty. There exists a* CPF *join expression from which Algorithm 2 derives a program whose cost is within a factor $r^2 + 3r - 6$ of the cost of an optimal join expression over $\mathcal{D}$, where r is the number of relations in $\mathcal{D}$.*

Example 3.2.5. From the join expression tree in Figure 16, Algorithm 2 generates a program consisting of 22 $(= 4^2 + 3 \cdot 4 - 6)$ statements. $\square$

The computation time of Algorithm 2 is polynomial in the number of relations and the number of attributes. Case 2 in Algorithm 2 first tests if $\mathbf{W}_i \cap \mathbf{W}_n \subseteq$

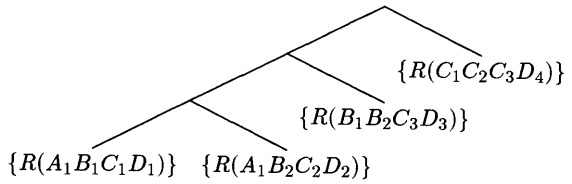


FIG. 16. Join expression tree over $\{R(A_1B_1C_1D_1), R(A_1B_2C_2D_2), R(B_1B_2C_3D_3), R(C_1C_2C_3D_4)\}$.

$\mathbf{V}_{n-1}$ for each $i = 0, \ldots, n-1$. Let a denote the number of attributes. Then each test needs $O(a \ln a)$ time. The number of those tests is maximum if the right child of any internal node in the input tree is a leaf. Thus, the number of tests is less than $r^2/2$, where r is the number of relations. In Theorem 3.2.3, we also have proved that the number of statements generated by Algorithm 2 is bounded by $r^2 + 3r - 6$. Consequently, the computation time of Algorithm 2 is polynomial in the number of relations and the number of attributes.

4. *Discussion*

To compute the join of some set of relations we have employed programs that use joins, semijoins, and projections. We have shown that there exists a CPF join expression tree from which we can derive a program whose cost is within a constant factor of the cost of an optimal join expression tree.

As the cost measure, we have used the sum of tuples appearing the input relations or the relations generated, because we assume that the actual best possible method, given in Ullman [1989], is used. Chandra and Merlin [1977] used matrix multiplication as an implementation to compute the join of relations and proved that there exists an implementation that is within a constant factor of the optimal solution. This result depends on choosing matrix multiplication, which could be significantly more expensive than the best possible method. If R_1 is the join of R_2 and R_3, R_i has a_i attributes and d is the number of all constants, the cost of executing "$R_1 := R_2 \bowtie R_3$" by matrix multiplication is $d^{a_1} + d^{a_2} + d^{a_3}$, which could be far greater than $|R_1| + |R_2| + |R_3|$.

Avoiding Cartesian products is one heuristic commonly used to reduce the search space for join expressions. We can further restrict the search space by using linear join expressions. The following question then naturally arises:

> Does there exist a linear and CPF join expression tree from which we can derive a program whose cost is within a constant factor of the cost of an optimal join expression tree ?

Our method cannot be generalized directly in order to solve the above question, because, as indicated in Example 3.1.1, Algorithm 1 does not necessarily produce a linear and CPF join expression tree.

If the scheme of these relations is acyclic (tree), we can compute the join in time polynomial in the size of the input relations and the output [Beeri et al. 1983; Bernstein and Goodman 1981; Ullman 1989]. The method first applies to the relations a full reducer [Bernstein and Goodman 1981] (a sequence of semijoins) which makes the relations globally consistent by eliminating dangling tuples. It then takes the join by using a monotone join expression (ordering of

joins) [Beeri et al. 1983], which guarantees that no intermediate join has more tuples than the final join. For cyclic schemes, however, the problem cannot be solved in polynomial time unless P = NP, because checking whether the join of a set of relations is nonempty is an NP-complete problem [Chandra and Merlin 1977].

Many attempts have been made to reduce cyclic problems to acyclic problems. Goodman and Shmueli [1984] showed that if we have a program for computing the join of some relations, the program implicitly creates an embedded acyclic database consisting of the input relations and some other relations produced during the execution. Sagiv and Shmueli [1993] generalized the result to the class of programs that may have semijoin loops. Those results motivated us to generate programs that use core relations, because if $R(\mathbf{X})$ is a core relation of $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_n)\}$, $\{R(\mathbf{X}), R(\mathbf{W}_0), \ldots, R(\mathbf{W}_n)\}$ is an acyclic database. In other words, Algorithm 2 explicitly uses acyclic databases to generate programs.

In Theorem 3.2.3, we assume that the join of the input database $\mathcal{D}$ is nonempty. This property guarantees that the join of any subset of $\mathcal{D}$ is nonempty, which is essential to establish Lemma 3.2.2 and Theorem 3.2.3. We therefore cannot drop the condition.

In this paper, we compare the cost of the program computed by Algorithm 2 to the cost of the optimal join expression. The reader might think that this comparison a bit unfair and might want to know what happens if the comparison is with the optimal program that may use Cartesian products. Put another way, the question is:

> Does there exist a CPF join expression tree from which Algorithm 2 produces a program whose cost is within a constant factor of the cost of an optimal program?

The answer is "No." In what follows, we give a counterexample. Consider the database $\{R(ABC), R(CDE), R(EFA)\}$ whose tuples are shown below.

$R(ABC)$	$R(CDE)$	$R(EFA)$
$(1, a, 1)$	$(1, a, 1)$	$(1, a, -1)$
$\vdots$	$\vdots$	$\vdots$
$(10^{2k}, a, 1)$	$(1, a, 10^{2k})$	$(10^{2k}, a, -1)$
$(-1, b, -1)$	$(-1, b, -1)$	$(-1, b, 1)$
$\vdots$	$\vdots$	$\vdots$
$(-1, b, -10^{2k})$	$(-10^{2k}, b, -1)$	$(-1, b, 10^{2k})$
(c, c, c)	(c, c, c)	(c, c, c)

We assume that $k \geq 1$. Select a join expression tree for $\{R(ABC), R(CDE), R(EFA)\}$ arbitrarily, and apply Algorithm 2 to the tree to obtain a program. We can see that the program computes the join of two relations in $\{R(ABC), R(CDE), R(EFA)\}$ or the projection of the join onto ACE as a temporary relation during the computation. For instance, if the program is produced from $(R(ABC) \bowtie R(CDE)) \bowtie R(EFA)$, it computes $\pi_{ACE}(R(ABC) \bowtie R(CDE))$ during the computation. If the program is derived from $R(ABC) \bowtie (R(CDE) \bowtie R(EFA))$, it computes $R(CDE) \bowtie R(EFA)$ during the execution. The size of

those temporary relations is greater than 10^{4k}, and therefore the cost of the optimal program produced by Algorithm 2 is greater than 10^{4k}. On the other hand, consider the following program P:

$$
\begin{aligned}
R(BF) &:= \pi_{BF}((\pi_{AB}R(ABC)) \bowtie (\pi_{AF}R(EFA))) \\
&\quad (= \{(a, b), (b, a), (c, c)\}) \\
R(BD) &:= \pi_{BD}((\pi_{BC}R(ABC)) \bowtie (\pi_{CD}R(CDE))) \\
&\quad (= \{(a, a), (b, b), (c, c)\}) \\
R(DF) &:= \pi_{DF}((\pi_{DE}R(CDE)) \bowtie (\pi_{EF}R(EFA))) \\
&\quad (= \{(a, a), (b, b), (c, c)\}) \\
R(BDF) &:= R(BF) \bowtie R(BD) \bowtie R(DF) \\
&\quad (= \{(c, c, c)\}) \\
R(ABCDEF) &:= ((R(BDF) \bowtie R(ABC)) \bowtie R(CDE)) \bowtie R(EFA) \\
&\quad (= \{(c, c, c, c, c, c)\})
\end{aligned}
$$

For readability, each statement consists of some simple statements. The above program computes the join of $\{R(ABC), R(CDE), R(EFA)\}$ for any instance of the database, because $R(BDF) \supseteq \pi_{BDF}(R(ABC) \bowtie R(CDE) \bowtie R(EFA))$, and the last statement takes the join of $R(ABC)$, $R(CDE)$, $R(EFA)$, and $R(BDF)$. The cost of the above program is less than 10^{2k+1}. Consequently, the cost of the optimal program generated by Algorithm 2 could be greater than the cost of the above program P by a factor of an arbitrarily large number.

The way in which P is constructed is completely different from the idea of Algorithm 2. Algorithm 2 produces programs that compute least core relations, while P does not produce any least core relation, but P generates $R(BF)$, $R(BD)$, $R(DF)$, and $R(BDF)$. Each of B, D, and F appears in only one of $\{R(ABC), R(CDE), R(EFA)\}$. This difference however does not imply that the way in which Algorithm 2 produces programs is wrong. Actually, for other instance of the database, P could be much more expensive than a program that Algorithm 2 produces. The following database gives such an example:

$R(ABC)$	$R(CDE)$	$R(EFA)$
$(a, 1, a)$	$(a, 1, a)$	$(a, 1, b)$
$\vdots$	$\vdots$	$\vdots$
$(a, 10^{2k}, a)$	$(a, 10^{2k}, a)$	$(a, 10^{2k}, b)$
$(b, -1, b)$	$(b, -1, b)$	$(b, -1, a)$
$\vdots$	$\vdots$	$\vdots$
$(b, -10^{2k}, b)$	$(b, -10^{2k}, b)$	$(b, -10^{2k}, a)$
(c, c, c)	(c, c, c)	(c, c, c)

The cost of P is greater than 10^{4k}, while the cost of the program generated from $(R(ABC) \bowtie R(CDE)) \bowtie R(EFA)$ by Algorithm 2 is less than 10^{2k+1}.

Lastly, we present an interesting research problem related to the above discussion.

Can we make an algorithm that generates from a CPF join expression tree, a program whose cost is within a constant factor of the cost of the optimal program?

Appendix A

We provide two ways of creating a sequence of statements for computing the least core relation of $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_{n-1}), R(\mathbf{W}_n)\}$ from the least core relation of $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_{n-1})\}$. Let $\mathbf{V}$ be the least core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_{n-1}\}$. If $\mathbf{V}$ is also the least core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_{n-1}, \mathbf{W}_n\}$, we use Proposition A.1. Otherwise, we employ Proposition A.2.

PROPOSITION A.1. *If* $\mathbf{V}$ *is a core scheme of* $\{\mathbf{W}_0, \ldots, \mathbf{W}_{n-1}, \mathbf{W}_n\}$,

$$\pi_V(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1})) \ltimes R(\mathbf{W}_n)$$

$$= \pi_{\mathbf{V}}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}) \bowtie R(\mathbf{W}_n)).$$

PROOF. It is enough to prove

$$\pi_V(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1})) \bowtie R(\mathbf{W}_n)$$
$$= \pi_{\mathbf{V} \cup \mathbf{W}_n}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}) \bowtie R(\mathbf{W}_n)). \quad (1)$$

Let μ be an arbitrary element in the left-hand side of Eq. (1). Since $\mu[\mathbf{V}]$ is a tuple in $\pi_{\mathbf{V}}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}))$, there exists a tuple $\nu \in R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1})$ such that $\mu[\mathbf{V}]$ and ν agree with each other. ν and $\mu[\mathbf{W}_n] \in R(\mathbf{W}_n)$ agree with each other, because the set of their common attributes, $(\mathbf{W}_0 \cup \cdots \cup \mathbf{W}_{n-1}) \cap \mathbf{W}_n$, is a subset of $\mathbf{V}$, and $\mu[\mathbf{V}]$ and ν agree with each other. Thus, μ is a subset of the right-hand side of Eq. (1).

On the other hand, let ρ be an arbitrary element in the right-hand side of Eq. (1). $\rho[\mathbf{V}]$ is in $\pi_{\mathbf{V}}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}))$, and $\rho[\mathbf{W}_n]$ is in $R(\mathbf{W}_n)$. Since $\rho[\mathbf{V}]$ agrees with $\rho[\mathbf{W}_n]$, ρ belongs to the left-hand side of Eq. (1). □

For example, since BC is a core scheme of $\{AB, BC, CD, CE\}$, from Proposition A.1, we have

$$\pi_{BC}(R(AB) \bowtie R(BC) \bowtie R(CD)) \ltimes R(CE)$$
$$= \pi_{BC}(R(AB) \bowtie R(BC) \bowtie R(CD) \bowtie R(CE)).$$

PROPOSITION A.2. *Let* $R(\mathbf{V}_{n-1})$ *be the least core relation of a connected database* $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_{n-1})\}$. *Suppose that* $\mathbf{V}_{n-1}$ *is not the least core scheme of* $\{\mathbf{W}_0, \ldots, \mathbf{W}_{n-1}, \mathbf{W}_n\}$. *Then there exists* $\mathbf{W}_j (0 \leq j < n)$ *such that* $(\mathbf{W}_j \cap \mathbf{W}_n) - \mathbf{V}_{n-1} \neq \phi$. *Figure A.1 shows the hypergraph for* $\{\mathbf{V}_{n-1}, \mathbf{W}_j, \mathbf{W}_n\}$.

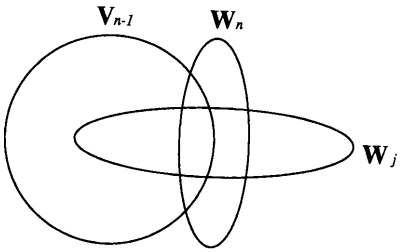


FIG. A.1. Hypergraph for $\{\mathbf{V}_{n-1}, \mathbf{W}_j, \mathbf{W}_n\}$ such that $(\mathbf{W}_j \cap \mathbf{W}_n) - \mathbf{V}_{n-1} \neq \phi$.

Let $\mathcal{F}$ be

$$\{R(\mathbf{W}_j) \mid 0 \leq j < n, \mathbf{W}_j \cap \mathbf{W}_n \neq \phi\}.$$

The following sequence of statements sets $R(\mathbf{V}_n)$ *to the least core relation of* $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_{n-1}), R(\mathbf{W}_n)\}$.

$$R(\mathbf{X}) := \pi_{\mathbf{V}_{n-1} \cap att(\mathcal{F})} R(\mathbf{V}_{n-1})$$

$$R(\mathbf{X}) := R(\mathbf{X}) \bowtie (\pi_{\mathbf{W}_j \cap (\mathbf{V}_{n-1} \cup \mathbf{W}_n)} R(\mathbf{W}_j)),$$

for each $R(\mathbf{W}_j) \in \mathcal{F}$ *such that* $(\mathbf{W}_j \cap \mathbf{W}_n) - \mathbf{V}_{n-1} \neq \phi$

$$R(\mathbf{X}) := R(\mathbf{X}) \ltimes R(\mathbf{W}_n)$$

$$R(\mathbf{V}_n) := R(\mathbf{V}_{n-1}) \bowtie R(\mathbf{X}).$$

Furthermore, throughout the execution,

$$|R(\mathbf{X})| \leq |\pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}))|.$$

PROOF. We give the proof later in this section. □

The reader might wonder why we have to make such a complex program for computing the least core relation of $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_{n-1}), R(\mathbf{W}_n)\}$. The reason is that we want to bound the size of the intermediate relation $R(\mathbf{X})$.

PROPOSITION A.3. *Let* $\mathbf{V}$ *be a core scheme of a connected database scheme* $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$. *Suppose that* $R(\mathbf{X})$ *stores* $\pi_{\mathbf{V}}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n))$. *For each* $\mathbf{W}_i$ *that is not a subset of* $\mathbf{V}$, *generate*

$$R(\mathbf{X}) := R(\mathbf{X}) \bowtie R(\mathbf{W}_i).$$

Then, throughout the execution of the above statements,

$$|R(\mathbf{X})| \leq |R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)|.$$

After the execution of the last statement,

$$R(\mathbf{X}) = R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n).$$

PROOF. We give the proof later in this section. □

In order to prove Proposition A.2 and Proposition A.3, we need the following three lemmas.

LEMMA A.4. *If* $\mathbf{V}$ *is a core scheme of* $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$,

$$\pi_{\mathbf{V}}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)) \bowtie R(\mathbf{W}_i)$$

$$= \pi_{\mathbf{V} \cup \mathbf{W}_i}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)). \tag{2}$$

When $R(\mathbf{X})$ *stores* $\pi_{\mathbf{V}}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n))$, *consider:*

$$R(\mathbf{X}) := R(\mathbf{X}) \bowtie R(\mathbf{W}_0)$$

$$R(\mathbf{X}) := R(\mathbf{X}) \bowtie R(\mathbf{W}_1)$$

$$\vdots$$

$$R(\mathbf{X}) := R(\mathbf{X}) \bowtie R(\mathbf{W}_m).$$

After the execution of the $(k + 1)$*-th statement,*

$$R(\mathbf{X}) = \pi_{\mathbf{V} \cup \mathbf{W}_0 \cup \ldots \cup \mathbf{W}_k}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)). \tag{3}$$

PROOF. We prove Eq. (2) first. Let μ be an arbitrary tuple of the left-hand side of Eq. (2). $\mu[\mathbf{V}]$ belongs to $\pi_{\mathbf{V}}(R\mathbf{W_0}) \bowtie \cdots \bowtie \mathbf{R}(\mathbf{W}_n))$. Since $\mathbf{V}$ is a core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$, for any $j \neq i$, $\mathbf{W}_i - \mathbf{V}$ and $\mathbf{W}_j - \mathbf{V}$ are disjoint. Thus, $R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)$ has a tuple ξ such that $\xi[\mathbf{W}_i] = \mu[\mathbf{W}_i]$ and $\xi[\mathbf{V}] = \mu[\mathbf{V}]$. Observe that μ is a projection of ξ onto $\mathbf{V} \cup \mathbf{W}_i$, and hence μ belongs to the right-hand side of Eq. (2).

On the other hand, let ρ be an arbitrary tuple of the right-hand side of Eq. (2). Note that $\rho[\mathbf{V}]$ is in $\pi_{\mathbf{V}}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n))$, and $\rho[\mathbf{W}_i]$ is in $R(\mathbf{W}_i)$. Since $\rho[\mathbf{V}]$ joins with $\rho[\mathbf{W}_i]$, ρ is an element of the left-hand side of Eq. (2).

Equation (3) is obtained by repeatedly applying Eq. (2). □

LEMMA A.5. *If* $\mathbf{V}$ *is a core scheme of* $\{\mathbf{W}_0, \ldots, \mathbf{W}_n\}$, *and* $\mathbf{U}_i$ *is a subset of* $\mathbf{W}_i$, *then*

$$\pi_{\mathbf{V}}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)) \bowtie \pi_{\mathbf{U}_i} R(\mathbf{W}_i)$$

$$= \pi_{\mathbf{V} \cup \mathbf{U}_i}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)). \tag{4}$$

When $R(\mathbf{X})$ *stores* $\pi_{\mathbf{V}}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n))$, *consider:*

$$R(\mathbf{X}) := R(\mathbf{X}) \bowtie \pi_{\mathbf{U}_0} R(\mathbf{W}_0)$$

$$R(\mathbf{X}) := R(\mathbf{X}) \bowtie \pi_{\mathbf{U}_1} R(\mathbf{W}_1)$$

$$\vdots$$

$$R(\mathbf{X}) := R(\mathbf{X}) \bowtie \pi_{\mathbf{U}_m} R(\mathbf{W}_m).$$

After the execution of the $(k + 1)$*-th statement,*

$$R(\mathbf{X}) = \pi_{\mathbf{V} \cup \mathbf{U}_0 \cup \cdots \cup \mathbf{U}_k}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)) \tag{5}$$

PROOF. Observe that

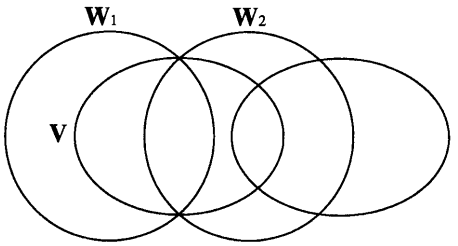


FIG. A.2. $\mathbf{V}$ is a core of $\{\mathbf{W}_1, \mathbf{W}_2\}$, and $\mathbf{W}_1 \cap \mathbf{W}_3 = \phi$.

$$R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)$$

$$= R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n) \bowtie \pi_{\mathbf{U}_i} R(\mathbf{W}_i) \tag{6}$$

Also note that $\mathbf{V}$ is a core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_n, \mathbf{U}_i\}$. From Lemma A.4, we have

$$\pi_{\mathbf{V}}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n) \bowtie \pi_{\mathbf{U}_i} R(\mathbf{W}_i)) \bowtie \pi_{\mathbf{U}_i} R(\mathbf{W}_i)$$

$$= \pi_{\mathbf{V} \cup \mathbf{U}_i}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n) \bowtie \pi_{\mathbf{U}_i} R(\mathbf{W}_i)). \tag{7}$$

Equation (6) and Eq. (7) imply Eq. (4).

Equation (5) is obtained by repeatedly applying Eq. (4). □

LEMMA A.6. *Let* $\mathbf{V}$ *be a core scheme of* $\{\mathbf{W}_1, \mathbf{W}_2\}$. *Suppose that* $\mathbf{W}_1 \cap \mathbf{W}_3 = \phi$. *Figure A.2 shows the hypergraph of* $\{\mathbf{V}, \mathbf{W}_1, \mathbf{W}_2, \mathbf{W}_3\}$. *Then we have*

$$\pi_{\mathbf{V}}(R(\mathbf{W}_1) \bowtie R(\mathbf{W}_2)) \bowtie \pi_{(\mathbf{V} \cup \mathbf{W}_3) \cap \mathbf{W}_2}(R(\mathbf{W}_1) \bowtie R(\mathbf{W}_2) \bowtie R(\mathbf{W}_3))$$

$$= \pi_{\mathbf{V} \cup (\mathbf{W}_2 \cap \mathbf{W}_3)}(R(\mathbf{W}_1) \bowtie R(\mathbf{W}_2) \bowtie R(\mathbf{W}_3)). \tag{8}$$

PROOF. Let ν be an arbitrary element in the left-hand side of Eq. (8). Note that ν is a tuple over $\mathbf{V} \cup (\mathbf{W}_2 \cap \mathbf{W}_3)$. As $\nu[(\mathbf{V} \cup \mathbf{W}_3) \cap \mathbf{W}_2]$ belongs to $\pi_{(\mathbf{V} \cup \mathbf{W}_3) \cap \mathbf{W}_2}(R(\mathbf{W}_1) \bowtie R(\mathbf{W}_2) \bowtie R(\mathbf{W}_3))$, there exist tuples $\omega_2 \in R(\mathbf{W}_2)$ and $\omega_3 \in R(\mathbf{W}_3)$ such that

$$\nu[(\mathbf{V} \cup \mathbf{W}_3) \cap \mathbf{W}_2],\ \omega_2 \text{ and } \omega_3 \text{ agree with each other.}$$

Furthermore, since $\nu[\mathbf{V}]$ is in $\pi_{\mathbf{V}}(R(\mathbf{W}_1) \bowtie R(\mathbf{W}_2))$, there exists a tuple $\omega_1 \in R(\mathbf{W}_1)$ such that

$$\nu[\mathbf{V}] \text{ and } \omega_1 \text{ agree with each other.}$$

We will prove each pair of $\{\nu, \omega_1, \omega_2, \omega_3\}$ agree with each other, which means that ν belongs to the right-hand side of Eq. (8).

—The set of common attributes of ν and ω_1 is $(\mathbf{V} \cup (\mathbf{W}_2 \cap \mathbf{W}_3)) \cap \mathbf{W}_1$, which is equal to $\mathbf{V} \cap \mathbf{W}_1$. As $\nu[\mathbf{V}]$ and $\omega_1 \in R(\mathbf{W}_1)$ agree with each other, ν and ω_1 also agree with each other.

—The set of common attributes of ν and ω_2 is $(\mathbf{V} \cup (\mathbf{W}_2 \cap \mathbf{W}_3)) \cap \mathbf{W}_2$, which is equal to $(\mathbf{V} \cup \mathbf{W}_3) \cap \mathbf{W}_2$. As $\nu[(\mathbf{V} \cup \mathbf{W}_3) \cap \mathbf{W}_2]$ and $\omega_2 \in R(\mathbf{W}_2)$ agree with each other, ν and ω_2 also agree with each other.

—The set of common attributes of ν and ω_3 is $(\mathbf{V} \cup (\mathbf{W}_2 \cap \mathbf{W}_3)) \cap \mathbf{W}_3$, which is equal to $\mathbf{W}_2 \cap \mathbf{W}_3$. $\nu[(\mathbf{V} \cup \mathbf{W}_3) \cap \mathbf{W}_2]$ and $\omega_3 \in R(\mathbf{W}_3)$ agree with each

other on their common attributes, $\mathbf{W}_2 \cap \mathbf{W}_3$. Thus, ν and ω_3 also agree with each other.

—The set of common attributes of ω_1 and ω_2 is $\mathbf{W}_1 \cap \mathbf{W}_2$. $\omega_1[\mathbf{W}_1 \cap \mathbf{W}_2] = \nu[\mathbf{W}_1 \cap \mathbf{W}_2]$, because $\nu[\mathbf{V}]$ and $\omega_1 \in R(\mathbf{W}_1)$ agree with each other, and $\mathbf{W}_1 \cap \mathbf{W}_2$ is a subset of $\mathbf{V} \cap \mathbf{W}_1$. Furthermore, $\nu[\mathbf{W}_1 \cap \mathbf{W}_2] = \omega_2[\mathbf{W}_1 \cap \mathbf{W}_2]$, because $\nu[(\mathbf{V} \cup \mathbf{W}_3) \cap \mathbf{W}_2]$ and $\omega_2 \in R(\mathbf{W}_2)$ agree with each other, and $\mathbf{W}_1 \cap \mathbf{W}_2$ is a subset of $(\mathbf{V} \cup \mathbf{W}_3) \cap \mathbf{W}_2$. Thus, $\omega_1[\mathbf{W}_1 \cap \mathbf{W}_2] = \omega_2[\mathbf{W}_1 \cap \mathbf{W}_2]$, and hence ω_1 and ω_2 agree with each other.

—It is trivial to show that ω_1 and ω_3 agree with each other, because the set of their common attributes is empty.

—ω_2 and ω_3 have been chosen so that they agree with each other.

On the other hand, let μ be an arbitrary element in the right-hand side of Eq. (8). $\mu[\mathbf{V}]$ belongs to $\pi_{\mathbf{V}}(R(\mathbf{W}_1) \bowtie R(\mathbf{W}_2) \bowtie R(\mathbf{W}_3))$, and hence it is also in $\pi_{\mathbf{V}}(R(\mathbf{W}_1) \bowtie R(\mathbf{W}_2))$. $\mu[(\mathbf{V} \cup \mathbf{W}_3) \cap \mathbf{W}_2]$ belongs to $\pi_{(\mathbf{V} \cup \mathbf{W}_3) \cap \mathbf{W}_2}(R(\mathbf{W}_1) \bowtie R(\mathbf{W}_2) \bowtie R(\mathbf{W}_3))$, because $(\mathbf{V} \cup \mathbf{W}_3) \cap \mathbf{W}_2$ is a subset of $\mathbf{V} \cup (\mathbf{W}_2 \cap \mathbf{W}_3)$. Thus, μ belongs to the left-hand side of Eq. (8). □

PROOF OF PROPOSITION A.2. Since $R(\mathbf{V}_{n-1})$ stores the least core relation of $\{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_{n-1})\}$,

$$R(\mathbf{V}_{n-1}) = \pi_{\mathbf{V}_{n-1}}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1})).$$

After the execution of "$R(\mathbf{X}) := \pi_{\mathbf{V}_{n-1} \cap att(\mathcal{F})} R(\mathbf{V}_{n-1})$,"

$$R(\mathbf{X}) = \pi_{\mathbf{V}_{n-1} \cap att(\mathcal{F})}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1})),$$

and thereby

$$|R(\mathbf{X})| \leq |\pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}))|.$$

Next we show that

$$(\mathbf{V}_{n-1} \cap att(\mathcal{F})) \cup$$
$$(\cup\{\mathbf{W}_j \cap (\mathbf{V}_{n-1} \cup \mathbf{W}_n) | R(\mathbf{W}_j) \in \mathcal{F}, (\mathbf{W}_j \cap \mathbf{W}_n) - \mathbf{V}_{n-1} \neq \phi\})$$
$$= (\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F}).$$

Let X be an arbitrary attribute in the left-hand side. If X is in $(\mathbf{V}_{n-1} \cap att(\mathcal{F}))$, X is trivially an element of the right-hand side. Otherwise, there exists $R(\mathbf{W}_j) \in \mathcal{F}$ such that X is in $\mathbf{W}_j \cap (\mathbf{V}_{n-1} \cup \mathbf{W}_n)$, and hence X is in the right-hand side. On the other hand, let Y be an arbitrary attribute in the right-hand side. If Y is in $\mathbf{V}_{n-1} \cap att(\mathcal{F})$, Y is trivially in the left-hand side. Otherwise, Y is in $\mathbf{W}_n \cap att(\mathcal{F})$, and therefore there exists some $\mathbf{W}_j$ such that Y is in $\mathbf{W}_j \cap \mathbf{W}_n$. As Y is not in $\mathbf{V}_{n-1} \cap att(\mathcal{F})$, Y must be in $(\mathbf{W}_j \cap \mathbf{W}_n) - \mathbf{V}_{n-1}$, and hence Y is in the left-hand side.

Then, from Lemma A.5, throughout the execution of

$$"R(\mathbf{X}) := R(\mathbf{X}) \bowtie (\pi_{\mathbf{W}_j \cap (\mathbf{V}_{n-1} \cup \mathbf{W}_n)} R(\mathbf{W}_j))"$$

for each $R(\mathbf{W}_j) \in \mathcal{F}$ such that $(\mathbf{W}_j \cap \mathbf{W}_n) - \mathbf{V}_{n-1} \neq \phi$, $R(\mathbf{X})$ is a projection of $\pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}))$. Thus, we have

$$|R(\mathbf{X})| \leq |\pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}))|.$$

Also from Lemma A.5, after the execution,

$$R(\mathbf{X}) = \pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1})).$$

$(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})$ is a core scheme of $\{\mathbf{W}_0 \cup \cdots \cup \mathbf{W}_{n-1}, \mathbf{W}_n\}$, because for each $\mathbf{W}_j (0 \leq j < n)$, if $R(\mathbf{W}_j) \in \mathcal{F}$, we have $\mathbf{W}_j \cap \mathbf{W}_n \subseteq \mathbf{W}_n \cap att(\mathcal{F})$, and otherwise $\mathbf{W}_j \cap \mathbf{W}_n = \phi$. Note that $R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1})$ is a relation over $\mathbf{W}_0 \cup \cdots \cup \mathbf{W}_{n-1}$. From Proposition A.1, after the execution of "$R(\mathbf{X}) := R(\mathbf{X}) \ltimes R(\mathbf{W}_n)$," we have

$$\begin{aligned} R(\mathbf{X}) &= \pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1})) \ltimes R(\mathbf{W}_n) \\ &= \pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}) \bowtie R(\mathbf{W}_n)) \end{aligned}$$

and hence

$$|R(\mathbf{X})| \leq |\pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}))|.$$

Finally, consider executing "$R(\mathbf{V}_n) := R(\mathbf{V}_{n-1}) \bowtie R(\mathbf{X})$." Let $\bar{\mathcal{F}}$ be $\{R(\mathbf{W}_j) \mid 0 \leq j < n, \mathbf{W}_j \cap \mathbf{W}_n = \phi\}$. Note that

$$\mathcal{F} \cup \bar{\mathcal{F}} = \{R(\mathbf{W}_0), \ldots, R(\mathbf{W}_{n-1})\},$$

and therefore

$$\begin{aligned} R(\mathbf{V}_{n-1}) &= \pi_{\mathbf{V}_{n-1}}((\bowtie \bar{\mathcal{F}}) \bowtie (\bowtie \mathcal{F})) \\ R(\mathbf{X}) &= \pi_{(\mathbf{V}_{n-1} \cup \mathbf{W}_n) \cap att(\mathcal{F})}((\bowtie \bar{\mathcal{F}}) \bowtie \quad (\bowtie \mathcal{F}) \bowtie R(\mathbf{W}_n)). \end{aligned}$$

$\bowtie \bar{\mathcal{F}}$ is a relation over $att(\bar{\mathcal{F}})$, and $\bowtie \mathcal{F}$ is a relation over $att(\mathcal{F})$. Since $\mathbf{V}_{n-1}$ is the least core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_{n-1}\}$, $\mathbf{V}_{n-1}$ is a core scheme of $\{att(\bar{\mathcal{F}}), att(\mathcal{F})\}$. From the definition of $\bar{\mathcal{F}}$, $att(\bar{\mathcal{F}}) \cap \mathbf{W}_n = \phi$. From Lemma A.6, we have

$$R(\mathbf{V}_{n-1}) \bowtie R(\mathbf{X}) = \pi_{\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathcal{F}))}((\bowtie \bar{\mathcal{F}}) \bowtie (\bowtie \mathcal{F}) \bowtie R(\mathbf{W}_n)).$$

Consequently

$$R(\mathbf{V}_n) = \pi_{\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathcal{F}))}(R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_{n-1}) \bowtie R(\mathbf{W}_n)).$$

$\mathbf{V}_{n-1} \cup (\mathbf{W}_n \cap att(\mathcal{F}))$ is the least core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_{n-1}, \mathbf{W}_n\}$, because $\mathbf{V}_{n-1}$ is the least core scheme of $\{\mathbf{W}_0, \ldots, \mathbf{W}_{n-1}\}$, and $\mathbf{W}_n \cap att(\mathcal{F})$ is the set of all attributes that appear in $\mathbf{W}_n$ and $\mathbf{W}_j$ $(0 \leq j \leq n - 1)$. □

PROOF OF PROPOSITION A.3. From Lemma A.4, $R(\mathbf{X})$ is always a projection of $R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)$, we have

$$|R(\mathbf{X})| \leq |R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)|.$$

Also from Lemma A.4, $R(\mathbf{X})$ is set to $R(\mathbf{W}_0) \bowtie \cdots \bowtie R(\mathbf{W}_n)$ after the last statement. □

ACKNOWLEDGMENTS. I thank Jeff Ullman for insightful examples that motivated me to pursue this work. On earlier versions of this paper, Y.C. Tay and Surajit Chaudhuri gave me valuable comments, which are included in Section 4. I also thank anonymous referees, whose helpful suggestions led to the revision of the examples in Section 2 and 3, and the inclusion of some problems in Section 4.

REFERENCES

BEERI, C. R., FAGIN, R., MAIER, D., AND YANNAKAKIS, M. 1983. On the desirability of acyclic database schemas. *J. ACM 30*, 3 (July), 479–513.

BERNSTEIN, P. A., AND GOODMAN, N. 1981. The power of natural semijoins. *SIAM J. Comput. 10*, 4 (Nov.) 751–771.

CHANDRA, A. K., AND MERLIN, P. M. 1977. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the 9th ACM Symposium on the Theory of Computing*, (Boulder, Colo., May 2–4). ACM, New York, pp. 77–90.

GOODMAN, N., AND SHMUELI, O. 1984. The tree projection theorem and relational query processing. *J. Comput. Syst. Sci. 28*, 1 (Feb.), 60–79.

SAGIV, Y., AND SHMUELI, O. 1993. Solving queries by tree projections. *ACM Trans. Database Syst. 18*, 3 (Sept.), 487–511.

SELINGER, P. G., ASTRAHAN, M. M., CHAMBERLIN, D. D., LORIE, P. A., AND PRICE, T. G. 1979. Access path selection in a relational database system. In *Proceedings of the 1979 ACM-SIGMOD International Conference on Management of Data* (Boston, Mass., May 30–June 1). ACM, New York, pp. 23–34.

SMITH, D. E., AND GENESERETH, M. R. 1985. Ordering conjunctive queries. *Artif. Intell*. 26, 171–215.

SWAMI, A., AND GUPTA, A. 1988. Optimization of large join queries. In *Proceedings of the 1988 ACM–SIGMOD International Conference on Management of Data* (Chicago, Ill., June 1–3). ACM, New York, 1988, pp. 8–17.

SWAMI, A. 1989. Optimization of large join queries: Combining heuristics and combinatorial techniques. In *Proceedings of the 1989 ACM–SIGMOD International Conference on Management of Data* (Portland, Ore., May 31–June 2). ACM, New York, 367–376.

TAY, Y. C. 1993. On the optimality of strategies for multiple joins. *J. ACM 40*, 5 (Nov.), 1067–1086.

ULLMAN, J. D. 1989. *Principles of Database and Knowledge-Base Systems*, vol. 2. Computer Science Press, New York.

WONG, E., AND YOUSEFFI, K. 1976. Decomposition–A strategy for query processing. *ACM Trans. Database Syst. 1*, 3, (Sept.), 223–241.

YANNAKAKIS, M. 1981. Algorithms for acyclic database schemes. In *Proceedings of the International Conference on Very Large Data Bases* (Cannes, France, Sept. 9–11). ACM, New York, pp. 82–94.

RECEIVED AUGUST 1994; REVISED AUGUST 1996; ACCEPTED MAY 1996