On the Optimality of Strategies for Multiple Joins

Y. C. TAY

National University of Singapore, Kent Ridge, Singapore

Abstract. Relational database systems rely on the join operator to assemble data for answering queries. Although the order of (natural) joins—here called the *strategy for computing the joins*—does not affect the final result, it does determine to a large extent the response time of the query. Query optimizers therefore try to pick an optimal strategy. In practice, optimizers usually restrict their search for an optimal strategy to strategies that are linear (e.g., of the form $((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$), or that avoid Cartesian products, or both.

The purpose of this paper is to examine the conditions under which an optimizer can find an optimal strategy, despite having restricted the scope of its search. Specifically, sufficient conditions are given under which (1) a linear strategy that is optimum will not use Cartesian products, (2) there is an optimum strategy that does not use Cartesian products, and (3) there is an optimum strategy that is linear and that does not use Cartesian products. (Optimality is with respect to the number of tuples generated by a strategy.) The necessity of these conditions is illustrated through examples.

The conditions do not assume uniformity in the distribution of attribute values, nor independence in the attributes. Instead, they are either a formalization of heuristic assumptions, or based on semantic constraints. For example, the conditions are satisfied if all join attributes form superkeys. The analytic framework can be adapted for database acyclicity, lossless joins, unions, and intersections.

Categories and Subject Descriptors: F.2.2 [**Theory of Computation**]: Nonnumerical Algorithms and Problems—*sequencing and scheduling*; H.2.4 [**Database Management**]: Systems—*query processing*

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Cartesian product, heuristic, intersection, join strategy, join tree, linear strategy, lossless join, optimality, query optimizer, superkey, union.

1. *Introduction*

Consider the problem of evaluating the expression $R_1 \bowtie R_2 \bowtie R_3 \bowtie R_4$, where R_1, R_2, R_3, and R_4 are relations. If we use parentheses to indicate the order of the evaluation, then there are 3 orderings (after renaming the relations) of the form $(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4)$ and 12 orderings of the form $((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$. Among these 15 possible orderings which is optimum? This question is

An abstract of this paper appears in *Proceedings of the 9th ACM Symposium on Principles of Database Systems* (Nashville, Tenn., Apr. 2–4). ACM, New York, 1990, pp. 110–119.

Author's address: Department of Mathematics, National University of Singapore, Kent Ridge 0511, Republic of Singapore.

of practical importance because evaluating the joins in the wrong order could produce an enormous number of intermediate tuples, even if the final result is small. The problem can be traced back to the early days of System R and INGRES, but there is also a renewed interest in the problem recently because of an expectation that nontraditional database systems may have to evaluate expressions containing hundreds of joins [12, 18, 22].

One heuristic that is common in the evaluation of multiple joins is the use of linear strategies, that is, strategies like $((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4$. Linear strategies have practical appeal because they can be programmed as nested loops, can take advantage of existing indices, and can use pipelining. Another common heuristic is to avoid Cartesian products; this is because Cartesian products generate spurious tuples that have to be eliminated eventually. Query optimizers in many well-known systems use one or both heuristics. For example, GAMMA [9] uses only linear strategies, INGRES [25] and Starburst [14] both avoid Cartesian products, and System R [20] and Office-by-Example [24] both use only linear strategies that avoid Cartesian products.

The purpose of the above heuristics is to restrict, for the sake of tractability, the query optimizer's search for a strategy to a small subspace of the entire strategy space. However, this restriction may also exclude the optimum strategy from the subspace. For example, experiments have shown [9] that for large queries, the cheapest linear strategy could be significantly more expensive than the cheapest possible (nonlinear) strategy.

The following question thus arises naturally: Under what conditions would be cheapest strategy in a searched subspace be optimum among all strategies? This is a variation of an open problem [21], and is the issue addressed by this paper. Specifically, we give conditions under which the following are true:

(1) A linear strategy that is optimum will not use Cartesian products.
(2) There is an optimum strategy that does not use Cartesian products.
(3) There is an optimum strategy that is linear and that does not use Cartesian products.

Phrased differently (but imprecisely), (1) says that the query optimizer can rule out the use of Cartesian products if it is searching within the subspace of linear strategies, while (2) says the optimizer can restrict the search for an optimum strategy to the subspace of strategies that do not use Cartesian products.

To the best of our knowledge, these results are the first of their kind. Previous work on optimum strategies for evaluating multiple joins were mostly on algorithms for finding the cheapest strategy in a selected subspace [11, 12, 20–22, 24, 25]. Our results complement such algorithms in the sense that we identify conditions that guarantee that the algorithms can find strategies that are optimum.

A second difference from previous work lies in the kind of assumptions underlying the analysis. Most work in the literature assume that attribute values are uniformly distributed for each attribute, and independently distributed for every set of attributes. These assumptions are generally believed to be unrealistic in practice, and known to be unsatisfactory in theory [4, 11]. In contrast, the assumptions we adopt are either a formalization of heuristic belief, or based on semantic constraints. For example, the aforementioned conditions are satisfied if joining attributes form superkeys of the relations.

A third break with previous work lies in our choice of cost measures. Instead of using a detailed cost model involving indices, page sizes, buffer allocation and so on, we simply use the number of tuples generated as the cost measure. This is partly necessitated by the analytic technique we use, partly motivated by the lack of unifying formulas for the cost of access paths and join methods [6, 27], and partly to provide results that are robust with respect to technological innovation. The last point refers to the fact that some previous formulas may be irrelevant in the context of parallel machines [16], large main memories [6], and novel secondary storage [5].

We begin in Section 2 (Preliminaries) with a formalization of the problem. The main results are derived in Section 3 (Results). Section 4 (Application) illustrates how the results are applicable in the presence of keys; we also give examples that demonstrate the necessity of the conditions imposed in the theorems. We conclude in Section 5 (Discussion) with an account of how the formalism can be adapted to consider database cyclicity, lossless joins, extension joins, (set) union, and intersection.

2. *Preliminaries*

Let $\mathbf{U}$ be a nonempty finite set of symbols, called *attributes*. A *relation scheme* is a nonempty subset of $\mathbf{U}$. For each attribute A, there is a set called the *domain* of A. A *tuple* over relation scheme $\mathbf{R}$ is a mapping that, for each $A \in \mathbf{R}$, associates an element of A's domain with A. A *relation state* over a relation scheme $\mathbf{R}$ is a finite set of tuples over $\mathbf{R}$; we usually use R to denote a relation state over $\mathbf{R}$. A *relation* is an ordered pair $(\mathbf{R}, R)$.

A *database scheme* is a finite nonempty set of relation schemes. A *database state* over a database scheme $\mathbf{D}$ is a set of relation states over $\mathbf{D}$'s relation schemes; we usually use D to denote a database state over $\mathbf{D}$. A *database* is an ordered pair $(\mathbf{D}, D)$.

If t is a tuple over $\mathbf{R}$ and $\mathbf{X} \subseteq \mathbf{R}$, $t[\mathbf{X}]$ is the restriction of the mapping to $\mathbf{X}$. If R and R' are relation states over $\mathbf{R}$ and $\mathbf{R}'$, respectively, the *natural join* of R and R' is

$$R \bowtie R' = \{t : t \text{ a tuple over } \mathbf{R} \cup \mathbf{R}', \text{ and } t[\mathbf{R}] \in R,\ t[\mathbf{R}'] \in R'\}.$$

For a quick grasp of the following terminology, one could imagine a database scheme as a graph with its relation schemes as nodes, and an edge between two nodes if and only if they have nonempty intersection. $\mathbf{D}_1$ is *linked* to $\mathbf{D}_2$ (and vice versa) if and only if $(\bigcup \mathbf{D}_1) \cap (\bigcup \mathbf{D}_2) \neq \phi$, where $\bigcup \mathbf{D} = \bigcup_{\mathbf{R} \in \mathbf{D}} \mathbf{R}$. For example, $\{ABC, BE, DF\}$ is linked to $\{CG, GH\}$ but $\{AB, BE, DF\}$ is not linked to $\{CG, GH\}$. (Here, A, B, C etc. are attributes, and ABC denotes the relation scheme $\{A, B, C\}$.)

$\mathbf{D}_1$ and $\mathbf{D}_2$ are *disjoint* if and only if $\mathbf{D}_1 \cap \mathbf{D}_2 = \phi$. Thus, $\{ABC, BE, DF\}$ and $\{CG, GH\}$ are disjoint, but $\{ABC, BE, CG, DF\}$ and $\{CG, GH\}$ are not disjoint.

A database scheme $\mathbf{D}$ is *unconnected* if it is the union of two (disjoint) database schemes that are not linked to each other; otherwise, it is *connected*. For example, $\{ABC, BE, DF\}$ is unconnected, but $\{ABC, BE, AF, DF\}$ is connected.

A *component* of $\mathbf{D}$ is a connected subset $\mathbf{D}'$ that is not linked to $\mathbf{D} - \mathbf{D}'$. Thus, $\{ABC, BE\}$ and $\{DF\}$ are the components of $\{ABC, BE, DF\}$. Note that, although $\{ABC, BE, DF\}$ is linked to $\{CG, GH\}$, their union—$\{ABC, BE, DF, CG, GH\}$—remains unconnected.

To *evaluate* a database $\mathscr{D}$ is to compute the natural join of $\mathscr{D}$'s relation states. By the commutativity and associativity of joins, the order of the joins does not affect the final result; however, the order determines the intermediate relation states that are computed, and thus the cost of the evaluation.

Informally—as in examples—we can use parentheses to indicate the order of the joins, and strings of attributes (e.g., ABC) to denote both the relation scheme and the relation state. In the results and proofs, however, we have to distinguish schemes from states, and keep track of the databases. This requires extra notation, and can be done less clumsily if we use binary trees instead of parentheses to indicate the order of the joins.

Let $\mathscr{D} = (\mathbf{D}, D)$ be a database. Let R_D denote the relation state $\bowtie_{R \in D} R$. A *strategy* S to evaluate $\mathscr{D}$ (or, simply, a strategy for $\mathscr{D}$) is a rooted binary tree such that

(S1) each node of S is an ordered pair $[\mathbf{D}', R_{D'}]$, where $\mathbf{D}' \subseteq \mathbf{D}$;
(S2) the root of S is $root(S) = [\mathbf{D}, R_D]$;
(S3) every internal node $[\mathbf{D}', R_{D'}]$ has two children $[\mathbf{D}_1, R_{D_1}]$ and $[\mathbf{D}_2, R_{D_2}]$ where $\mathbf{D}_1$ and $\mathbf{D}_2$ are disjoint and $\mathbf{D}' = \mathbf{D}_1 \cup \mathbf{D}_2$ (it follows that $R_{D'} = R_{D_1} \bowtie R_{D_2}$);
(S4) the leaves are of the form $[\{\mathbf{R}\}, R]$, where $\mathbf{R} \in \mathbf{D}$.

The internal nodes of S are called *steps* of S. If the children of a step are $[\mathbf{D}_1, R_{D_1}]$ and $[\mathbf{D}_2, R_{D_2}]$, we usually refer to that step as $[\mathbf{D}_1, R_{D_1}] \bowtie [\mathbf{D}_2, R_{D_2}]$. Intuitively, a strategy indicates the partial order for the evaluation of the joins in $\bowtie_{R \in D} R$. It is clear from (S1)–(S4) that a subtree S' of S is a strategy for $(\mathbf{D}', D')$, where $root(S') = [\mathbf{D}', R_{D'}]$; S' is called a *substrategy* of S.

Since a binary tree with n leaves has $n - 1$ internal nodes, a strategy for $(\mathbf{D}, D)$ therefore has $|\mathbf{D}| - 1$ steps. If $\mathbf{D} = \{\mathbf{R}\}$, there is only one strategy for $\mathscr{D}$, namely the one with $[\{\mathbf{R}\}, R]$ as the only node; we call this a *trivial* strategy. Thus, a *linear* strategy is one where every step has a trivial strategy as a child.

We say a step $[\mathbf{D}_1, R_{D_1}] \bowtie [\mathbf{D}_2, R_{D_2}]$ *uses a Cartesian product* if and only if $\mathbf{D}_1$ and $\mathbf{D}_2$ are not linked to each other. A strategy S *uses Cartesian products* if and only if it has a step that uses a Cartesian product. For example, the strategy $(ABC \bowtie DF) \bowtie BCD$ uses a Cartesian product.

We say S evaluates $\mathscr{D}$'s components *individually* if and only if for each component $\mathbf{E}$ of $\mathbf{D}$, $[\mathbf{E}, R_E]$ is a step in S. Thus, the strategy $(ABC \bowtie BE) \bowtie DF$ evaluates the components of $\{ABC, BE, DF\}$ individually, but the strategy $(ABC \bowtie DF) \bowtie BE$ does not.

In practice, query optimizers usually avoid Cartesian products: if $\mathbf{D}$ is connected, only strategies that do not use Cartesian products are considered; if $\mathbf{D}$ is unconnected, an extension of this heuristic would mean that the components are evaluated individually, and Cartesian products are not used for each of the components. Formally, S *avoids Cartesian products* if and only if S evaluates $\mathscr{D}$'s components individually and has exactly $comp(\mathbf{D}) - 1$ steps that use Cartesian products, where $comp(\mathbf{D})$ is the number of components in $\mathscr{D}$. (Every strategy must necessarily use at least $comp(\mathbf{D}) - 1$ Cartesian products.) For example, the strategy $((ABC \bowtie BE) \bowtie (CG \bowtie GH)) \bowtie DF$ avoids cartesian products, but $((ABC \bowtie CG) \bowtie (BE \bowtie GH)) \bowtie DF$ does not (although the latter evaluates components individually).

We define the cost $\tau(S)$ of a strategy in terms of the number of tuples in the intermediate and final relation states generated in the evaluation. Let $\tau(R)$ be the number of tuples in R, for any relation state R. We extend τ to steps by defining $\tau([\mathbf{D}', R_{D'}]) = \tau(R_{D'})$. Thus, if $s = [\mathbf{D}_1, R_{D_1}] \bowtie [\mathbf{D}_2, R_{D_2}]$ is a step, then $\tau(s) = \tau(R_{D_1} \bowtie R_{D_2})$; also, $\tau(R_{D_1} \bowtie R_{D_2}) \leq \tau(R_{D_1})\tau(R_{D_2})$ for any R_{D_1} and R_{D_2}, where equality holds if s uses a Cartesian product. We further extend τ to strategies by defining $\tau(S) = \sum_{i=1}^{k-1}\tau(s_i)$, where $k = |\mathbf{D}|$ and $s_1, \ldots, s_{k-1}$ are the steps of S. A strategy S is *τ-optimum* if and only if $\tau(S) \leq \tau(S')$ for all strategies S' that evaluate the same database as S.

By the finiteness of databases, τ-optimum strategies always exist. Also, if S is a τ-optimum strategy for $(\mathbf{D}, D)$ and S' is a substrategy of S, say *root*$(S') = [\mathbf{D}', R_{D'}]$, then S' must be τ-optimum for $(\mathbf{D}', D')$; otherwise, we can replace S' by a τ-optimum strategy S'' for $(\mathbf{D}', D')$ and thus obtain a new strategy that contradicts the τ-optimality of S. The replacement here consists of plucking the subtree S', and grafting on S'' in its place. We now formally define these two operations.

Let S be a strategy, $s = [\mathbf{D}', R_{D'}] \bowtie [\mathbf{D}'', R_{D''}]$ a step of S, and $S_{D'}$ and $S_{D''}$ the substrategies such that *root*$(S_{D'}) = [\mathbf{D}', R_{D'}]$ and *root*$(S_{D''}) = [\mathbf{D}'', R_{D''}]$ (see Figure 1).

Transform S into a new tree T as follows:

(i) if $[\mathbf{E}, R_E]$ is an ancestor of s in S, replace it by $[\mathbf{E} - \mathbf{D}'', R_{E-D''}]$;
(ii) replace the subtree rooted at s by $S_{D'}$.

It is easy to see that the resulting T is a strategy for $(\mathbf{D} - \mathbf{D}'', D - D'')$. We say T is obtained from S by *plucking* $S_{D''}$. Conversely, let S and $S_{D''}$ be strategies for $(\mathbf{D}, D)$ and $(\mathbf{D}'', D'')$, respectively, where $\mathbf{D}$ and $\mathbf{D}''$ are disjoint. Let $S_{D'}$ be a substrategy of S (see Figure 2). Transform S into a new tree T as follows:

(i) if $[\mathbf{E}, R_E]$ is an ancestor of *root*$(S_{D'})$ in S, replace it by $[\mathbf{E} \cup \mathbf{D}'', R_{E \cup D''}]$;
(ii) replace $S_{D'}$ by a new subtree whose root has $S_{D'}$ and $S_{D''}$ as subtrees.

The resulting T is a strategy for $(\mathbf{D} \cup \mathbf{D}'', D \cup D'')$. We say T is obtained from S by *grafting* $S_{D''}$ *above* $S_{D'}$.

3. *Results*

Strategies that avoid Cartesian products are favored by query optimizers; this is partly based on a heuristic belief that a join on one or more attributes will not produce more tuples than a Cartesian product. The following condition is a formalization of this assumption:

$C_1(\mathscr{D})$: Suppose $\mathscr{D} = (\mathbf{D}, D)$. For all disjoint subsets $\mathbf{E}$, $\mathbf{E}_1$ and $\mathbf{E}_2$ of $\mathbf{D}$, if each of them is connected and $\mathbf{E}$ is linked to $\mathbf{E}_1$ but not to $\mathbf{E}_2$, then $\tau(R_E \bowtie R_{E_1}) \leq \tau(R_E \bowtie R_{E_2})$.

Although this condition is a heuristic assumption, we will see in Section 4 how it can be satisfied if the relations have keys.

It is clear that if $\mathscr{D}$ satisfies $C_1(\mathscr{D})$, then $\mathscr{D}' = (\mathbf{D}', \mathscr{D}')$ also satisfies $C_1(\mathscr{D}')$ for any $\mathbf{D}' \subseteq \mathbf{D}$. Our first lemma shows that, essentially, C_1 does not require connectedness of $\mathbf{E}$ and $\mathbf{E}_2$.

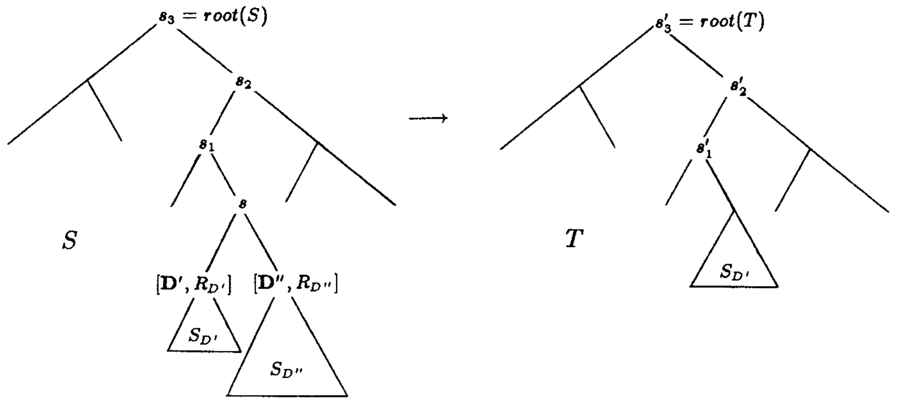

FIG. 1. Plucking $S_{D''}$.

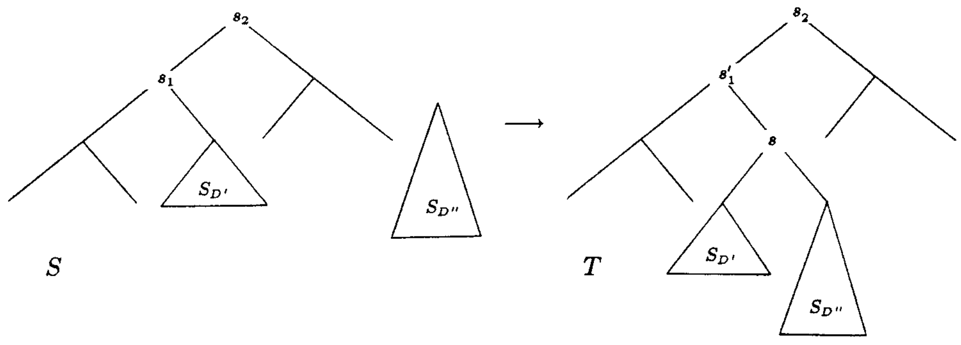

FIG. 2. Grafting $S_{D''}$ above $S_{D'}$.

LEMMA 1. *Let $\mathscr{D} = (\mathbf{D}, D)$ be a database satisfying $C_1(\mathscr{D})$, and $R_D \neq \phi$. Then for all disjoint subsets $\mathbf{E}$, $\mathbf{E}_1$ and $\mathbf{E}_2$ of $\mathbf{D}$, if $\mathbf{E}_1$ is connected and $\mathbf{E}$ is linked to $\mathbf{E}_1$ but not to $\mathbf{E}_2$, we have $\tau(R_E \bowtie R_{E_1}) \leq \tau(R_E \bowtie R_{E_2})$.*

PROOF

Case 1. Suppose $\mathbf{E}_2$ is connected. If $\mathbf{E}$ is connected, then the claim is just $C_1(\mathscr{D})$, so suppose $\mathbf{E}$ is unconnected. Let $\mathbf{F}$ be a component of $\mathbf{E}$ such that $\mathbf{F}$ is linked to $\mathbf{E}_1$. By $C_1(\mathscr{D})$, we have $\tau(R_F \bowtie R_{E_1}) \leq \tau(R_F \bowtie R_{E_2})$. It follows that

$$\tau(R_E \bowtie R_{E_1}) = \tau(R_F \bowtie R_{E-F} \bowtie R_{E_1}) \leq \tau(R_F \bowtie R_{E_1})\tau(R_{E-F})$$
$$\leq \tau(R_F \bowtie R_{E_2})\tau(R_{E-F}) = \tau(R_F \bowtie R_{E_2} \bowtie R_{E-F}),$$

since $\mathbf{E} - \mathbf{F}$ is not linked to $\mathbf{F} \cup \mathbf{E}_2$. Therefore, $\tau(R_E \bowtie R_{E_1}) \leq \tau(R_E \bowtie R_{E_2})$.

Case 2. Suppose $\mathbf{E}_2$ is unconnected. Let $\mathbf{F}$ be a component of $\mathbf{E}_2$. Since $\mathbf{E}$ is not linked to $\mathbf{F}$, we have

$$\begin{aligned} \tau(R_E \bowtie R_{E_1}) &\leq \tau(R_E \bowtie R_F) && \text{from Case 1} \\ &\leq \tau(R_E \bowtie R_F)\tau(R_{E_2-F}) && \text{since } R_D \neq \phi \text{ implies } \tau(R_{E_2-F}) \geq 1 \\ &= \tau(R_E \bowtie R_F \bowtie R_{E_2-F}) && \text{since } \mathbf{E}_2 - \mathbf{F} \text{ is not linked to } \mathbf{E} \cup \mathbf{F}, \end{aligned}$$

so $\tau(R_E \bowtie R_{E_1}) \leq \tau(R_E \bowtie R_{E_2})$. □

The condition C_1 ensures that avoiding Cartesian products at each step of the evaluation will always be locally optimum; however, the resulting strategy may not be globally optimum, as the following example shows.

Example 1. Let $\mathbf{R}_1 = AB$, $\mathbf{R}_2 = BC$, $\mathbf{R}_3 = DE$, and $\mathbf{R}_4 = FG$. Suppose

$$R_1 = \{(p,0),(q,0),(r,0),(s,1)\} \qquad \text{and}$$
$$R_2 = \{(0,w),(0,x),(0,y),(1,z)\},$$

so $\tau(R_1) = \tau(R_2) = 4$ and $\tau(R_1 \bowtie R_2) = 10$, and suppose $\tau(R_3) = \tau(R_4) = 7$. One can verify that this database satisfies C_1.

There are three strategies that avoid Cartesian products, namely

$$((R_1 \bowtie R_2) \bowtie R_3) \bowtie R_4,$$
$$((R_1 \bowtie R_2) \bowtie R_4) \bowtie R_3,$$

and

$$(R_1 \bowtie R_2) \bowtie (R_3 \bowtie R_4).$$

If these strategies are S_1, S_2, and S_3, respectively, then $\tau(S_1) = \tau(S_2) = 10 + 70 + 490 = 570$ and $\tau(S_3) = 10 + 49 + 490 = 549$. They are not τ-optimum, because if S_4 is the strategy $(R_1 \bowtie R_3) \bowtie (R_2 \bowtie R_4)$, then $\tau(S_4) = 28 + 28 + 490 = 546$, which is less than the other three, so the τ-optimum strategy does not avoid Cartesian products. □

However, a slight strengthening of the condition C_1 is sufficient to prove that a linear strategy that is τ-optimum must necessarily avoid Cartesian products. The stronger version of C_1 is as follows:

$C_1'(\mathscr{D})$: Suppose $\mathscr{D} = (\mathbf{D}, D)$. For all disjoint subsets $\mathbf{E}$, $\mathbf{E}_1$ and $\mathbf{E}_2$ of $\mathbf{D}$, if each of them is connected and $\mathbf{E}$ is linked to $\mathbf{E}_1$ but not to $\mathbf{E}_2$, then $\tau(R_E \bowtie R_{E_1}) < \tau(R_E \bowtie R_{E_2})$.

This condition has a consequence analogous to Lemma 1.

LEMMA 1′. *Let $\mathscr{D} = (\mathbf{D}, D)$ be a database satisfying $C'_1(\mathscr{D})$, and $R_D \neq \phi$. Then, for all disjoint subsets $\mathbf{E}$, $\mathbf{E}_1$ and $\mathbf{E}_2$ of $\mathbf{D}$, if $\mathbf{E}_1$ is connected and $\mathbf{E}$ is linked to $\mathbf{E}_1$ but not to $\mathbf{E}_2$, we have $\tau(R_E \bowtie R_{E_1}) < \tau(R_E \bowtie R_{E_2})$.*

PROOF. The proof is similar to that for Lemma 1. □

THEOREM 1. *Let $\mathscr{D} = (\mathbf{D}, D)$ be a database where $\mathbf{D}$ is connected and $R_D \neq \phi$. If $\mathscr{D}$ satisfies $C_1'(\mathscr{D})$, and S is a linear strategy for $\mathscr{D}$ that is τ-optimum, then S does not use Cartesian products.*

PROOF. Suppose S uses Cartesian products. Let s be the last step in S to use a Cartesian product, that is, all ancestors of s in S do not use Cartesian products. (Note that s cannot be the root since $\mathbf{D}$ is connected.) Since S is linear, let the children of s be $[\mathbf{E}, R_E]$ and $[\{R'\}, R']$, and the parent of s be $s' = s \bowtie [\{R''\}, R'']$ (see Figure 3).

By the definition of s, $\{\mathbf{R}''\}$ must be linked to $\mathbf{E} \cup \{\mathbf{R}'\}$, so either $\{\mathbf{R}'\}$ is linked to $\{\mathbf{R}''\}$ or $\mathbf{E}$ is linked to $\{\mathbf{R}''\}$.

Case 1. Suppose $\{\mathbf{R}'\}$ is linked to $\{\mathbf{R}''\}$. Then transform S into another strategy T_1 by plucking the trivial substrategy for $(\{\mathbf{R}'\},\{R'\})$ and grafting it

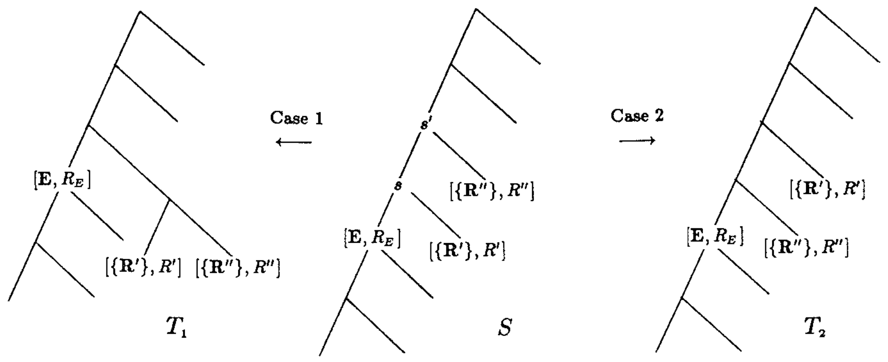

FIG. 3. Transforming S into T_1 and T_2

above the trivial substrategy for $(\{\mathbf{R}''\}, \{R''\})$ (see Figure 3). Since $\{\mathbf{R}'\}$ is not linked to $\mathbf{E}$ (s uses a Cartesian product), we get $\tau(R' \bowtie R'') < \tau(R' \bowtie R_E)$ by Lemma 1′. Now,

$$\tau(S) - \tau(T_1) = \tau(R' \bowtie R_E) - \tau(R' \bowtie R'') > 0,$$

so $\tau(T_1) < \tau(S)$. This contradicts the assumption that S is τ-optimum.

Case 2. Suppose $\mathbf{E}$ is linked to $\{\mathbf{R}''\}$. Then transform S into another strategy T_2 by exchanging the positions of $[\{\mathbf{R}'\}, R']$ and $[\{\mathbf{R}''\}, R'']$ in S (see Figure 3). Since $\mathbf{E}$ is not linked to $\{\mathbf{R}'\}$, we have $\tau(R_E \bowtie R'') < \tau(R_E \bowtie R')$ by Lemma 1′. Thus,

$$\tau(S) - \tau(T_2) = \tau(R_E \bowtie R') - \tau(R_E \bowtie R'') > 0,$$

so $\tau(T_2) < \tau(S)$, again contradicting the τ-optimality of S.

We conclude that s does not exist, that is, S does not use Cartesian products. □

In other words, Theorem 1 says that under the given conditions, a linear strategy that uses Cartesian products cannot be τ-optimum.

Note that we require $\mathbf{D}$ to be connected, since this is the case that is of practical interest. Also, if $R_D = \phi$, then the evaluation of the database can be abandoned as soon as an intermediate relation state is null, so we require $R_D \neq \phi$. The condition C_1' in the theorem cannot be relaxed to C_1; this is demonstrated in Example 3 (Section 4). We now return to condition C_1.

LEMMA 2. *Let $\mathscr{D} = (\mathbf{D}, D)$ be a database satisfying $C_1(\mathscr{D})$ with $R_D \neq \phi$, and S a strategy for $\mathscr{D}$. Suppose root(S) has children $[\mathbf{D}_1, R_{D_1}]$ and $[\mathbf{D}_2, R_{D_2}]$, where $\mathbf{D}_1$ is connected, $\mathbf{D}_2$ is unconnected, $\mathbf{D}_1$ is linked to $\mathbf{D}_2$, and the substrategy for $(\mathbf{D}_2, D_2)$ evaluates its components individually. Then there is a strategy S' for $\mathscr{D}$ such that $\tau(S') \leq \tau(S)$, and root(S') has two children $[\mathbf{D}_1', R_{D_1'}]$ and $[\mathbf{D}_2', R_{D_2'}]$ such that $\mathbf{D}_1'$ is connected, and comp($\mathbf{D}_1'$) + comp($\mathbf{D}_2'$) < comp($\mathbf{D}_1$) + comp($\mathbf{D}_2$).*

PROOF. Let S_{D_1} and S_{D_2} be the substrategies of S for $(\mathbf{D}_1, D_1)$ and $(\mathbf{D}_2, D_2)$, respectively. Since $\mathbf{D}_1$ is linked to $\mathbf{D}_2$, we may assume $\mathbf{D}_1$ is linked to some component $\mathbf{E}$ of $\mathbf{D}_2$. Since S_{D_2} evaluates its components individually, $[\mathbf{E}, R_E]$ is a step of S_{D_2}. Let S_E be the substrategy of S rooted at $[\mathbf{E}, R_E]$.

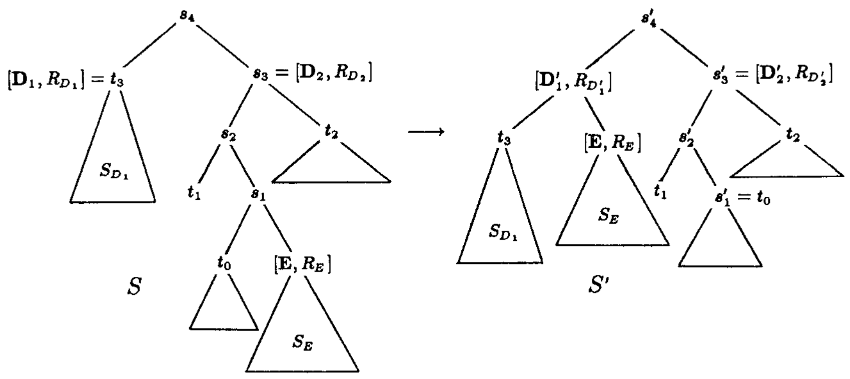

FIG. 4. Plucking S_E and grafting it above S_{D_1}.

Obtain from S a strategy S' for $\mathscr{D}$ by plucking S_E and grafting it above S_{D_1} (see Figure 4).

Suppose the parent of $[\mathbf{E}, R_E]$ in S is $s_1 = [\mathbf{E}, R_E] \bowtie t_0$. Let the parent of s_i in S be $s_{i+1} = s_i \bowtie t_i$ for $i = 1, 2, \ldots, m-1$, where $s_m = \mathit{root}(S)$ (so $t_{m-1} = [\mathbf{D}_1, R_{D_1}]$ and $s_{m-1} = [\mathbf{D}_2, R_{D_2}]$). Similarly, let s'_1 in S' be t_0, and the parent of s'_i in S' be s'_{i+1} for $i = 1, 2, \ldots, m-1$ (so $s'_m = \mathit{root}(S')$). Then, $s'_i \bowtie [\mathbf{E}, R_E] = s_i$ and $s'_{i+1} = s'_i \bowtie t_i$ for $i = 1, \ldots, m-1$. Now for $i = 1, \ldots, m-2$,

$$\begin{aligned}
\tau(s_{i+1}) &= \tau(s_i \bowtie t_i) = \tau(s'_i \bowtie [\mathbf{E}, R_E] \bowtie t_i) \\
&= \tau([\mathbf{E}, R_E])\tau(s'_i \bowtie t_i) \qquad \text{since } \mathbf{E} \text{ is a component of } \mathbf{D}_2 \\
&= \tau(R_E)\tau(s'_{i+1}) \\
&\geq \tau(s'_{i+1}) \qquad \text{since } R_D \neq \phi \text{ implies } \tau(R_E) \geq 1
\end{aligned}$$

so

$$\begin{aligned}
\tau(S') &= \tau(S) - (\tau(s_1) + \cdots + \tau(s_m)) + \tau\big([\mathbf{D}_1, R_{D_1}] \bowtie [\mathbf{E}, R_E]\big) \\
&\quad + \tau(s'_2) + \cdots + \tau(s'_m) \\
&= \tau(S) - \tau([\mathbf{E}, R_E] \bowtie t_0) + \tau\big([\mathbf{E}, R_E] \bowtie [\mathbf{D}_1, R_{D_1}]\big) \\
&\quad + \sum_{i=1}^{m-2} (\tau(s'_{i+1}) - \tau(s_{i+1})) \\
&\qquad\qquad \text{since } \tau(s_m) = \tau(s'_m) \\
&\leq \tau(S) - \tau([\mathbf{E}, R_E] \bowtie t_0) + \tau\big([\mathbf{E}, R_E] \bowtie [\mathbf{D}_1, R_{D_1}]\big).
\end{aligned}$$

Since $\mathbf{E}$ is linked to $\mathbf{D}_1, \mathbf{D}_1$ is connected, and $s_1 = [\mathbf{E}, R_E] \bowtie t_0$ uses a Cartesian product, we have (by Lemma 1) $\tau([\mathbf{E}, R_E] \bowtie [\mathbf{D}_1, R_{D_1}]) \leq \tau([\mathbf{E}, R_E] \bowtie t_0)$, so $\tau(S') \leq \tau(S)$.

Let $\mathbf{D}'_1 = \mathbf{D}_1 \cup \mathbf{E}$ and $\mathbf{D}'_2 = \mathbf{D}_2 - \mathbf{E}$. Then $\mathit{root}(S')$ have $[\mathbf{D}'_1, R_{D'_1}]$ and $[\mathbf{D}'_2, R_{D'_2}]$ as children, where $\mathbf{D}'_1$ is connected because $\mathbf{D}_1$ and $\mathbf{E}$ are each connected, and they are linked to each other. Moreover, $\mathbf{E}$ is a component of $\mathbf{D}_2$, so $\mathit{comp}(\mathbf{D}'_2) = \mathit{comp}(\mathbf{D}_2) - 1$, and

$$\mathit{comp}(\mathbf{D}'_1) + \mathit{comp}(\mathbf{D}'_2) = 1 + \mathit{comp}(\mathbf{D}_2) - 1 < \mathit{comp}(\mathbf{D}_1) + \mathit{comp}(\mathbf{D}_2). \quad \square$$

The above consequence of C_1 is useful for transforming strategies (from S to S') while maintaining τ-optimality. When C_1 is augmented with the following condition C_2, a τ-optimum strategy would be drawn into the subspace of strategies that avoid Cartesian products.

$C_2(\mathscr{D})$: Let $\mathscr{D} = (\mathbf{D}, D)$. For all disjoint subsets $\mathbf{E}_1$ and $\mathbf{E}_2$ of $\mathbf{D}$, if $\mathbf{E}_1$ is connected, $\mathbf{E}_2$ is connected, and $\mathbf{E}_1$ is linked to $\mathbf{E}_2$, then $\tau(R_{E_1} \bowtie R_{E_2}) \leq \tau(R_{E_1})$ or $\tau(R_{E_1} \bowtie R_{E_2}) \leq \tau(R_{E_2})$.

Example 2. Example 1 shows that C_1 does not imply C_2, since $\tau(R_1 \bowtie R_2) = 10$, which is larger than $\tau(R_1)$ and $\tau(R_2)$.

Now suppose $\mathbf{R}'_1 = AB$, $\mathbf{R}'_2 = BC$ and $\mathbf{R}'_3 = DE$. Further, let

$$R'_1 = \{(1,x),(2,y),(3,y),(4,y),(5,y),(6,y),(7,y),(8,y)\}$$

and

$$R'_2 = \{(y,0),(u,0),(v,0)\},$$

so $\tau(R'_1) = 8$, $\tau(R'_2) = 3$, and $\tau(R'_1 \bowtie R'_2) = 7$, and let $\tau(R'_3) = 2$. Then $\tau(R'_1 \bowtie R'_2) < \tau(R'_1)$, so C_2 is satisfied. However, C_1 is not satisfied, since $\tau(R'_2 \bowtie R'_1) > 6 = \tau(R'_2 \bowtie R'_3)$. Thus, C_2 does not imply C_1.

We conclude that C_1 and C_2 are independent. □

LEMMA 3. *Let $\mathscr{D} = (\mathbf{D}, D)$ be a database satisfying $C_1(\mathscr{D})$ and $C_2(\mathscr{D})$, with $R_D \neq \phi$, and S a strategy for $\mathscr{D}$. Suppose root(S) has children $[\mathbf{D}_1, R_{D_1}]$ and $[\mathbf{D}_2, R_{D_2}]$, where $\mathbf{D}_1$ is unconnected, $\mathbf{D}_2$ is unconnected, $\mathbf{D}_1$ is linked to $\mathbf{D}_2$, and the substrategies for $(\mathbf{D}_1, D_1)$ and $(\mathbf{D}_2, D_2)$ evaluate their components individually. Then there is a strategy S' for $\mathscr{D}$ such that $\tau(S') \leq \tau(S)$, and root(S') has two children $[\mathbf{D}'_1, R_{D'_1}]$ and $[\mathbf{D}'_2, R_{D'_2}]$ such that comp$(\mathbf{D}'_1)$ + comp$(\mathbf{D}'_2)$ < comp$(\mathbf{D}_1)$ + comp$(\mathbf{D}_2)$.*

PROOF. Since $\mathbf{D}_1$ is linked to $\mathbf{D}_2$, $\mathbf{D}_1$ must have a component $\mathbf{E}_1$ that is linked to a component $\mathbf{E}_2$ of $\mathbf{D}_2$. By $C_2(\mathscr{D})$, we may assume

$$\tau(R_{E_1} \bowtie R_{E_2}) \leq \tau(R_{E_1}). \tag{1}$$

If $\mathbf{E}_1$ is a component of some $\mathbf{F} \subseteq \mathbf{D}$, then

$$\begin{aligned}\tau(R_F \bowtie R_{E_2}) &= \tau(R_{E_1} \bowtie R_{E_2} \bowtie R_{F-E_1})\\ &\leq \tau(R_{E_1} \bowtie R_{E_2})\tau(R_{F-E_1})\\ &\leq \tau(R_{E_1})\tau(R_{F-E_1}) \qquad \text{by (1)}\\ &= \tau(R_F)\end{aligned}$$

since $\mathbf{E}_1$ is a component of $\mathbf{F}$. (2)

Since the substrategies for $(\mathbf{D}_1, D_1)$ and $(\mathbf{D}_2, D_2)$ evaluate their components individually, $[\mathbf{E}_1, R_{E_1}]$ and $[\mathbf{E}_2, R_{E_2}]$ must be steps in S, so let S_{E_1} and S_{E_2} be the substrategies rooted at $[\mathbf{E}_1, R_{E_1}]$ and $[\mathbf{E}_2, R_{E_2}]$, respectively. Obtain a strategy S' for $\mathscr{D}$ from S by plucking S_{E_2} and grafting it above S_{E_1} (see Figure 5).

Let u_1 be the parent of $[\mathbf{E}_1, R_{E_1}]$ in S, and u_{i+1} the parent of u_i in S for $i = 1, \ldots, m-1$, where u_m = root(S). Suppose the parent of $[\mathbf{E}_2, R_{E_2}]$ in S is

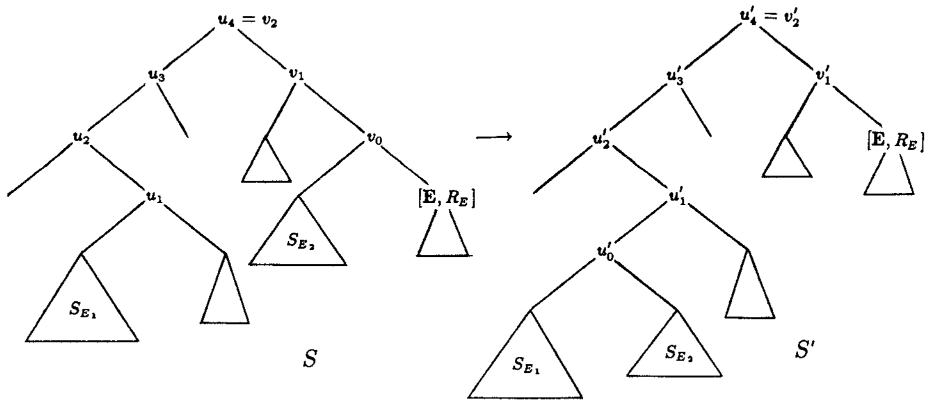

FIG. 5. Plucking S_{E_2} and grafting it above S_{E_1}.

$v_0 = [\mathbf{E}_2, R_{E_2}] \bowtie [\mathbf{E}, R_E]$; let v_{i+1} be the parent of v_i in S for $i = 0, \ldots, n-1$, where $v_n = root(S)$. Now let $u'_0 = [\mathbf{E}_1, R_{E_1}] \bowtie [\mathbf{E}_2, R_{E_2}]$ and u'_{i+1} be the parent of u'_i in S' for $i = 1, \ldots, m-1$; let v'_1 be the parent of $[\mathbf{E}, R_E]$ in S', and v'_{i+1} the parent of v'_i in S' for $i = 1, \ldots, n-1$.

Since $\mathbf{E}_1$ is a component of $\mathbf{D}_1$, and therefore of the database schemes in u_i for $i \le m-1$, we have, from (2),

$$\tau(u'_i) = \tau\left(u_i \bowtie \left[\mathbf{E}_2, R_{E_2}\right]\right) \le \tau(u_i) \qquad \text{for} \quad i = 1, \ldots, m-1. \tag{3}$$

On the other hand, $\mathbf{E}_2$ is a component of $\mathbf{D}_2$, so

$$\tau(v_i) = \tau\left(v'_i \bowtie \left[\mathbf{E}_2, R_{E_2}\right]\right) = \tau(v'_i)\tau\left(R_{E_2}\right) \ge \tau(v'_i) \qquad \text{for} \quad i = 1, \ldots, n-1, \tag{4}$$

since $R_D \ne \phi$ implies $\tau(R_{E_2}) \ge 1$. Therefore

$$\begin{aligned}
\tau(S) - \tau(S') &= (\tau(u_1) + \cdots + \tau(u_m) + \tau(v_0) + \cdots + \tau(v_{n-1})) \\
&\quad - (\tau(u'_0) + \cdots + \tau(u'_m) + \tau(v'_1) + \cdots + \tau(v'_{n-1})) \\
&\ge -\tau(u'_0) + \tau(v_0) \quad \text{by (3), (4) and } u_m = u'_m = [\mathbf{D}, R_D] \\
&= \tau\left(R_{E_2} \bowtie R_E\right) - \tau\left(R_{E_2} \bowtie R_{E_1}\right). \qquad (5)
\end{aligned}$$

Since $\mathbf{E}_1$ is connected and $\mathbf{E}_2$ is linked to $\mathbf{E}_1$ but not to $\mathbf{E}$ (recall $\mathbf{E}_2$ is a component of $\mathbf{D}_2$), it follows from (5) and Lemma 1 that $\tau(S) - \tau(S') \ge 0$, that is, $\tau(S') \le \tau(S)$.

Let $\mathbf{D}'_1 = \mathbf{D}_1 \cup \mathbf{E}_2$ and $\mathbf{D}'_2 = \mathbf{D}_2 - \mathbf{E}_2$, so $[\mathbf{D}'_1, R_{D'_1}]$ and $[\mathbf{D}'_2, R_{D'_2}]$ are children of $root(S')$. Then $comp(\mathbf{D}'_2) = comp(\mathbf{D}_2) - 1$ because $\mathbf{E}_2$ is a component of $\mathbf{D}_2$; moreover, $comp(\mathbf{D}'_1) \le comp(\mathbf{D}_1)$, so $comp(\mathbf{D}'_1) + comp(\mathbf{D}'_2) < comp(\mathbf{D}_1) + comp(\mathbf{D}_2)$. □

Although we are ultimately interested only in connected database schemes, we need the following lemma for unconnected schemes in the inductive proof for Theorem 2.

LEMMA 4. *Let $\mathscr{D} = (\mathbf{D}, D)$ be a database satisfying $C_1(\mathscr{D})$ and $C_2(\mathscr{D})$, with $R_D \neq \phi$. There is a τ-optimum strategy for $\mathscr{D}$ that evaluates its components individually.*

PROOF. We prove the claim by induction on $|\mathbf{D}|$. If $|\mathbf{D}| = 1$, the claim is vacuously true, so suppose the claim is true whenever $|\mathbf{D}| < k$, for some $k \geq 2$.

Consider any $\mathscr{D}_0 = (\mathbf{D}_0, D_0)$ satisfying $C_1(\mathscr{D}_0)$ and $C_2(\mathscr{D}_0)$, $R_{D_0} \neq \phi$, and $|\mathbf{D}_0| = k$. Let S be a τ-optimum strategy for $\mathscr{D}_0$, and $[\mathbf{D}_1, R_{D_1}]$ and $[\mathbf{D}_2, R_{D_2}]$ the children of *root*(S). Then, the substrategies for $(\mathbf{D}_1, D_1)$ and $(\mathbf{D}_2, D_2)$ must themselves be τ-optimum, and we may assume (by the induction hypothesis) that they evaluate the components of $(\mathbf{D}_1, D_1)$ and $(\mathbf{D}_2, D_2)$ individually. If $\mathbf{D}_1$ is not linked to $\mathbf{D}_2$, then S evaluates the components of $\mathscr{D}$ individually, as claimed.

Suppose now that $\mathbf{D}_1$ is linked to $\mathbf{D}_2$. If $\mathbf{D}_0$ is connected, then S trivially evaluates $\mathscr{D}_0$'s components (only one) individually.

If $\mathscr{D}_0$ is unconnected, then either $\mathbf{D}_1$ or $\mathbf{D}_2$ must be unconnected (since $\mathbf{D}_1$ is linked to $\mathbf{D}_2$), so suppose $\mathbf{D}_2$ is unconnected. By Lemmas 2 and 3, there is a strategy S' for $\mathscr{D}_0$ such that $\tau(S') \leq \tau(S)$, and *root*(S') have children $[\mathbf{D}'_1, R_{D'_1}]$ and $[\mathbf{D}'_2, R_{D'_2}]$ such that *comp*$(\mathbf{D}'_1)$ + *comp*$(\mathbf{D}'_2)$ < *comp*$(\mathbf{D}_1)$ + *comp*$(\mathbf{D}_2)$. Since S is τ-optimum, this S' is also τ-optimum. Applying the induction hypothesis (on the children of *root*(S')) and Lemmas 2 and 3 repeatedly in this manner, we must eventually get a τ-optimum strategy T for $\mathscr{D}_0$ such that the children of *root*(T) have between them the least number of components possible, namely *comp*$(\mathbf{D}_0)$. That happens when $\mathbf{E}_1$ is not linked to $\mathbf{E}_2$, where *root*(T) = $[\mathbf{E}_1, R_{E_1}] \bowtie [\mathbf{E}_2, R_{E_2}]$, so the components of $\mathbf{D}_0$ are also components of $\mathbf{E}_1$ and $\mathbf{E}_2$. By the induction hypothesis, we may assume the substrategies for $(\mathbf{E}_1, E_1)$ and $(\mathbf{E}_2, E_2)$ evaluate their components individually, so T evaluates the components of $\mathscr{D}_0$ individually. □

Here is the result that says, given C_1 and C_2, the query optimizer can restrict the search for an optimum strategy to the subspace of strategies that do not use Cartesian products:

THEOREM 2. *Let $\mathscr{D} = (\mathbf{D}, D)$ be a database where $\mathbf{D}$ is connected and $R_D \neq \phi$. If $\mathscr{D}$ satisfies $C_1(\mathscr{D})$ and $C_2(\mathscr{D})$, then there is a τ-optimum strategy for $\mathscr{D}$ that does not use Cartesian products.*

PROOF. The proof is by induction on $|\mathbf{D}|$. The claim is vacuously true for $|\mathbf{D}| = 1$, so suppose it is true whenever $|\mathbf{D}| < k$, for some $k \geq 2$. Consider any $\mathscr{D}_0 = (\mathbf{D}_0, D_0)$ satisfying $C_1(\mathscr{D}_0)$ and $C_2(\mathscr{D}_0)$, $\mathbf{D}_0$ connected, $R_{D_0} \neq \phi$, and $|\mathbf{D}_0| = k$. Let S be a τ-optimum strategy for $\mathscr{D}_0$, and $[\mathbf{D}_1, R_{D_1}]$ and $[\mathbf{D}_2, R_{D_2}]$ the children of *root*(S). Since $\mathscr{D}_0$ is connected, $\mathbf{D}_1$ must be linked to $\mathbf{D}_2$.

Case 1. If $\mathbf{D}_1$ is connected and $\mathbf{D}_2$ is connected, then by the induction hypothesis and the fact that the substrategies for $(\mathbf{D}_1, D_1)$ and $(\mathbf{D}_2, D_2)$ must also be τ-optimum, we may assume these substrategies do not use Cartesian products. It follows that S does not use Cartesian products.

Case 2. If $\mathbf{D}_1$ is connected and $\mathbf{D}_2$ is unconnected, we may assume that the substrategy for $(\mathbf{D}_2, D_2)$ evaluates its components individually, since we can always replace it by a τ-optimum strategy that does (Lemma 4). By repeated application of Lemma 2 and replacing (where necessary) substrategies with ones that evaluate components individually, we conclude that there is a τ-optimum strategy S' for $\mathscr{D}_0$ such that both children of $root(S')$ have connected database schemes. By Case 1, the desired strategy exists for $\mathscr{D}_0$.

Case 3. If $\mathbf{D}_1$ is unconnected and $\mathbf{D}_2$ is unconnected, we may again assume the substrategies for $(\mathbf{D}_1, D_1)$ and $(\mathbf{D}_2, D_2)$ evaluate their components individually. By repeated application of Lemma 3, and replacing (where necessary) substrategies with ones that evaluate their components individually, we get a τ-optimum strategy for $\mathscr{D}_0$ such that the root has a child with a connected database scheme. By Cases 1 and 2, the desired strategy for $\mathscr{D}_0$ exists. □

It follows from Lemma 4 and Theorem 2 that, for any $\mathscr{D} = (\mathbf{D}, D)$ satisfying $C_1(\mathscr{D})$, $C_2(\mathscr{D})$ and $R_D \neq \phi$, there is a τ-optimum strategy for $\mathscr{D}$ that avoids Cartesian products.

The τ-optimum strategy in Theorem 2 may not be linear. However, we can transform it into a linear strategy if we strengthen the condition from C_2 to C_3, as follows:

$C_3(\mathscr{D})$: Let $\mathscr{D} = (\mathbf{D}, D)$. For all disjoint subsets $\mathbf{E}_1$ and $\mathbf{E}_2$ of $\mathbf{D}$, if $\mathbf{E}_1$ is connected, $\mathbf{E}_2$ is connected, and $\mathbf{E}_1$ is linked to $\mathbf{E}_2$, then $\tau(R_{E_1} \bowtie R_{E_2}) \leq \tau(R_{E_1})$ and $\tau(R_{E_1} \bowtie R_{E_2}) \leq \tau(R_{E_2})$.

In contrast to C_1 and C_2, C_1 and C_3 are not independent.

LEMMA 5. *Let $\mathscr{D} = (\mathbf{D}, D)$ be a database such that $R_D \neq \phi$. Then $C_3(\mathscr{D})$ implies $C_1(\mathscr{D})$.*

PROOF. Let $\mathbf{E}$, $\mathbf{E}_1$, and $\mathbf{E}_2$ be disjoint subsets of $\mathbf{D}$, each of them connected, and $\mathbf{E}$ linked to $\mathbf{E}_1$ but not to $\mathbf{E}_2$. Then

$$\begin{aligned} \tau(R_E \bowtie R_{E_1}) &\leq \tau(R_E) && \text{by } C_3(\mathscr{D}) \\ &\leq \tau(R_E)\tau(R_{E_2}) && \text{since } R_D \neq \phi \text{ implies } \tau(R_{E_2}) \geq 1 \\ &= \tau(R_E \bowtie R_{E_2}) && \text{since } \mathbf{E} \text{ is not linked to } \mathbf{E}_2, \end{aligned}$$

so $\mathscr{D}$ satisfies $C_1(\mathscr{D})$. □

Although C_3 implies C_1 and C_2, Example 5 (Section 4) shows that the converse is false.

In the next lemma, we say (for convenience) a strategy is *connected* if and only if it does not use Cartesian products. Further, we say a strategy S for database $\mathscr{D}$ is τ-optimum *among* connected strategies if and only if S is connected and $\tau(S) \leq \tau(S')$ for all connected strategies S' for $\mathscr{D}$.

LEMMA 6. *Let $\mathscr{D}$ be a database with a connected database scheme. If $\mathscr{D}$ satisfies $C_3(\mathscr{D})$, then there is a linear strategy for $\mathscr{D}$ that does not use Cartesian products, and that is τ-optimum among strategies that do not use Cartesian products.*

PROOF. The claim is that there is a linear connected strategy for $\mathscr{D}$ that is τ-optimum among connected strategies.

Let $\mathscr{D} = (\mathbf{D}, D)$. The proof is by induction on $|\mathbf{D}|$. The claim is obviously true for $|\mathbf{D}| = 2$. Suppose now that the claim is true whenever $|\mathbf{D}| < k$, for some $k \geq 3$. Consider any database $\mathscr{D}_0 = (\mathbf{D}_0, D_0)$ that satisfies $C_3(\mathscr{D}_0)$, where $\mathbf{D}_0$ is connected and $|\mathbf{D}_0| = k$.

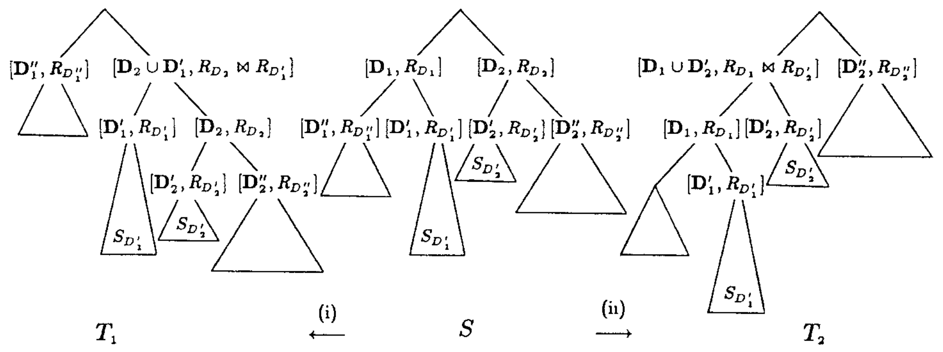
$[\mathbf{D}_1'', R_{D_1''}]$
$[\mathbf{D}_2 \cup \mathbf{D}_1', R_{D_2} \bowtie R_{D_1'}]$
$[\mathbf{D}_1', R_{D_1'}]$
$[\mathbf{D}_2, R_{D_2}]$
$[\mathbf{D}_2', R_{D_2'}]$
$[\mathbf{D}_2'', R_{D_2''}]$
$S_{D_1'}$
$S_{D_2'}$
$[\mathbf{D}_1, R_{D_1}]$
$[\mathbf{D}_2, R_{D_2}]$
$[\mathbf{D}_1'', R_{D_1''}]$
$[\mathbf{D}_1', R_{D_1'}]$
$[\mathbf{D}_2', R_{D_2'}]$
$[\mathbf{D}_2'', R_{D_2''}]$
$S_{D_1'}$
$S_{D_2'}$
$[\mathbf{D}_1 \cup \mathbf{D}_2', R_{D_1} \bowtie R_{D_2'}]$
$[\mathbf{D}_2'', R_{D_2''}]$
$[\mathbf{D}_1, R_{D_1}]$
$[\mathbf{D}_2', R_{D_2'}]$
$[\mathbf{D}_1', R_{D_1'}]$
$S_{D_2'}$
$S_{D_1'}$
T_1
(i)
S
(ii)
T_2

FIG. 6. Alternatives to S.

Let S be a connected strategy for $\mathscr{D}_0$ that is τ-optimum among connected strategies, and let the children of *root*(S) be $[\mathbf{D}_1, R_{D_1}]$ and $[\mathbf{D}_2, R_{D_2}]$. Since $\mathbf{D}_0$ is connected, $\mathbf{D}_1$ must be linked to $\mathbf{D}_2$.

Case 1. Suppose $|\mathbf{D}_1| = 1$ or $|\mathbf{D}_2| = 1$, say $|\mathbf{D}_1| = 1$, and S_{D_2} is the substrategy for $(\mathbf{D}_2, D_2)$. Since S is connected, S_{D_2} is also connected, so (by the induction hypothesis) there is a linear connected strategy S'_{D_2} for $(\mathbf{D}_2, D_2)$ that is τ-optimum among connected strategies. Since S_{D_2} must be τ-optimum among connected strategies, we have $\tau(S_{D_2}) = \tau(S'_{D_2})$. Replacing S_{D_2} by S'_{D_2} in S, we thus get a linear connected strategy S' for $\mathscr{D}_0$ that is τ-optimum among connected strategies.

Case 2. Suppose $|\mathbf{D}_1| > 1$ and $|\mathbf{D}_2| > 1$. Let the children of $[\mathbf{D}_1, R_{D_1}]$ in S be $[\mathbf{D}'_1, R_{D'_1}]$ and $[\mathbf{D}''_1, R_{D''_1}]$, and the children of $[\mathbf{D}_2, R_{D_2}]$ in S be $[\mathbf{D}'_2, R_{D'_2}]$ and $[\mathbf{D}''_2, R_{D''_2}]$ (see Figure 6).

Since $\mathbf{D}_1$ is linked to $\mathbf{D}_2$, we may assume $\mathbf{D}'_1$ is linked to $\mathbf{D}'_2$. Since S is a connected strategy, $\mathbf{D}_1$, $\mathbf{D}_2$, $\mathbf{D}'_1$, and $\mathbf{D}'_2$ are each connected.

Let S_{D_1}, S_{D_2}, $S_{D'_1}$, and $S_{D'_2}$ be the substrategies of S rooted at $[\mathbf{D}_1, R_{D_1}]$, $[\mathbf{D}_2, R_{D_2}]$, $[\mathbf{D}'_1, R_{D'_1}]$ and $[\mathbf{D}'_2, R_{D'_2}]$, respectively, and consider the following alternatives to S (see Figure 6):

(i) T_1 obtained from S by plucking $S_{D'_1}$ and grafting it above S_{D_2}, and
(ii) T_2 obtained from S by plucking $S_{D'_2}$ and grafting it above S_{D_1}.

Then,

$$\tau(T_1) - \tau(S) = \tau(R_{D'_1} \bowtie R_{D_2}) - \tau(R_{D_1}) \quad \text{and}$$
$$\tau(T_2) - \tau(S) = \tau(R_{D_1} \bowtie R_{D'_2}) - \tau(R_{D_2}). \tag{6}$$

Both T_1 and T_2 are connected strategies, so $\tau(S) \le \tau(T_1)$ and $\tau(S) \le \tau(T_2)$. It follows that

$$\tau(R_{D_1}) \le \tau(R_{D'_1} \bowtie R_{D_2}) \quad \text{and} \quad \tau(R_{D_2}) \le \tau(R_{D_1} \bowtie R_{D'_2}).$$

If one of these two inequalities is strict, then together with $\tau(R_{D'_1} \bowtie R_{D_2}) \le \tau(R_{D_2})$ from $C_3(\mathscr{D}_0)$, we get $\tau(R_{D_1}) < \tau(R_{D_1} \bowtie R_{D'_2})$, contradicting $C_3(\mathscr{D}_0)$. We conclude that

$$\tau(R_{D_1}) = \tau(R_{D'_1} \bowtie R_{D_2}) \quad \text{and} \quad \tau(R_{D_2}) = \tau(R_{D_1} \bowtie R_{D'_2}),$$

so $\tau(T_1) = \tau(S) = \tau(T_2)$ from (6), that is, T_1 and T_2 are also τ-optimum among connected strategies.

We can thus transfer a substrategy from either child of the root to the other while maintaining τ-optimality among connected strategies. By repeatedly transferring in one direction, we eventually get a strategy whose root has a child with a trivial substrategy. The result now follows from Case 1. □

THEOREM 3. *Let $\mathscr{D} = (\mathbf{D}, D)$ be a database where* $\mathbf{D}$ *is connected and $R_D \neq \phi$. If $\mathscr{D}$ satisfies $C_3(\mathscr{D})$, then there is a τ-optimum strategy for $\mathscr{D}$ that is linear and that does not use Cartesian products.*

PROOF. Since C_3 implies C_1 (Lemma 5) and C_2, $\mathscr{D}$ therefore satisfies the conditions of Theorem 2, so there is a τ-optimum strategy S for $\mathscr{D}$ that does not use Cartesian products. Moreover, from Lemma 6, there is a linear strategy S' for $\mathscr{D}$ that does not use Cartesian products, and that is τ-optimum among strategies that do not use Cartesian products. Therefore, $\tau(S') \leq \tau(S)$. Since S is τ-optimum (among all strategies), we get $\tau(S) \leq \tau(S')$, so $\tau(S') = \tau(S)$. In other words, we have a linear strategy S' for $\mathscr{D}$ that does not use Cartesian products, and that is τ-optimum. □

4. *Application*

Although condition C_1 is a formal statement of a common assumption, conditions C_2 and C_3 are motivated by semantic constraints on data. We now illustrate how C_2 and C_3 (and hence C_1, by Lemma 5) can be satisfied in the presence of such constraints. For the following, we adopt the standard terminology [23].

Suppose the only semantic constraints are functional dependencies, and the database $\mathscr{D} = (\mathbf{D}, D)$ has no nontrivial lossy joins; there is a polynomial algorithm for determining whether $\mathscr{D}$ has this property [1]. Now let $\mathbf{E}_1$ and $\mathbf{E}_2$ be disjoint subsets of $\mathbf{D}$, $\mathbf{E}_1$ connected, $\mathbf{E}_2$ connected, and $\mathbf{E}_1$ linked to $\mathbf{E}_2$, as in $C_2(\mathscr{D})$. Then $\mathbf{E}_1$ and $\mathbf{E}_2$ are lossless joins, and so is $\mathbf{E}_1 \cup \mathbf{E}_2$ (because $\mathbf{E}_1$ is linked to $\mathbf{E}_2$). It follows [17] that $\mathbf{R}_{E_1} \cap \mathbf{R}_{E_2}$ is a superkey of $\mathbf{R}_{E_1}$ or of $\mathbf{R}_{E_2}$, where $\mathbf{R}_{E_i} = \bigcup_{\mathbf{R} \in \mathbf{E}_i} \mathbf{R}$. We therefore have $\tau(R_{E_1} \bowtie R_{E_2}) \leq \tau(R_{E_2})$ (in the first case) or $\tau(R_{E_1} \bowtie R_{E_2}) \leq \tau(R_{E_1})$ (in the second case). $\mathscr{D}$ thus satisfies $C_2(\mathscr{D})$.

As for C_3, suppose $\mathscr{D} = (\mathbf{D}, D)$ is a database such that all joins are on superkeys, that is, if $\mathbf{R}_1, \mathbf{R}_2 \in \mathbf{D}$ and $\mathbf{R}_1 \cap \mathbf{R}_2 \neq \phi$, then $\mathbf{R}_1 \cap \mathbf{R}_2$ is a superkey of $\mathbf{R}_1$ and of $\mathbf{R}_2$. Observe first that if $\mathbf{K}$ is a superkey of $\mathbf{R}_1$, and $\mathbf{R}_1 \cap \mathbf{R}_2 \neq \phi$, then $\mathbf{K}$ is a superkey of $\mathbf{R}_1 \cup \mathbf{R}_2$. By induction, if $\mathbf{E} \subseteq \mathbf{D}$ and $\mathbf{E}$ is connected, then any superkey of a relation scheme in $\mathbf{E}$ is also a superkey of $\mathbf{R}_E$. Now let $\mathbf{E}_1$ and $\mathbf{E}_2$ be disjoint subsets of $\mathbf{D}$, $\mathbf{E}_1$ connected, $\mathbf{E}_2$ connected, and $\mathbf{E}_1$ linked to $\mathbf{E}_2$, as in $C_3(\mathscr{D})$. Then there are $\mathbf{R}_1 \in \mathbf{E}_1$ and $\mathbf{R}_2 \in \mathbf{E}_2$ such that $\mathbf{R}_1 \cap \mathbf{R}_2 \neq \phi$. By assumption, $\mathbf{R}_1 \cap \mathbf{R}_2$ is a superkey of $\mathbf{R}_1$ and therefore of $\mathbf{R}_{E_1}$, since $\mathbf{E}_1$ is connected. But $\mathbf{R}_1 \cap \mathbf{R}_2 \subseteq \mathbf{R}_{E_1} \cap \mathbf{R}_{E_2}$, so $\mathbf{R}_{E_1} \cap \mathbf{R}_{E_2}$ is a superkey of $\mathbf{R}_{E_1}$ and thus $\tau(R_{E_1} \bowtie R_{E_2}) \leq \tau(R_{E_2})$. By symmetry, $\tau(R_{E_1} \bowtie R_{E_2}) \leq \tau(R_{E_1})$, so $\mathscr{D}$ satisfies $C_3(\mathscr{D})$.

Are the conditions too restrictive? We give here three examples to illustrate the necessity of the sufficient conditions in the theorems.

Example 3 (*Theorem* 1). Consider a database with three relations that record the sports or games (G) that students (S) participate in, the courses (C) they take, and the laboratories (L) that courses use. Suppose we have a query that requires the join of these relations (say, "Do athletes avoid courses requiring laboratory work?"), and the database state is as follows:

Game	Student
Hockey	Mokhtar
Tennis	Lin

Student	Course
Mokhtar	Phy101
Mokhtar	Lang22
Lin	Lit101
Lin	Hist103
Katina	Psch123
Katina	Phy101
Sundram	Phy101
Sundram	Hist103

Course	Laboratory
Phy101	Fermi
Lang22	Chomsky

All three possible strategies generate the same number (4) of intermediate tuples, so all are τ-optimum. In particular, $(GS \bowtie CL) \bowtie SC$ is linear and τ-optimum, although it uses a Cartesian product. One can check that the database violates C_1', so Theorem 1 is not applicable; however, it satisfies C_1. The sufficient condition in Theorem 1 is therefore necessary, in the sense that C_1' cannot be relaxed to C_1. □

Example 4 (Theorem 2). Consider again the above setting, but with a different database state, as follows:

Game	Student
Hockey	Mokhtar
Tennis	Mokhtar
Tennis	Lin

Student	Course
Mokhtar	Lang22
Mokhtar	Lit104
Mokhtar	Phy101
Lin	Phy101
Lin	Hist103
Lin	Psch123
Katina	Lang22
Katina	Lit104
Katina	Phy101
Sundram	Phy101
Sundram	Lang22
Sundram	Hist103

Course	Laboratory
Phy101	Fermi
Lang22	Chomsky

Let the three possible strategies be $S_1 : (GS \bowtie SC) \bowtie CL$, $S_2 : GS \bowtie (SC \bowtie CL)$, and $S_3 : (GS \bowtie CL) \bowtie SC$. Then $\tau(S_1) = 9 + 5 = 14$, $\tau(S_2) = 7 + 5 = 12$, and $\tau(S_3) = 6 + 5 = 11$, so S_3 is τ-optimum, although it uses a Cartesian product. The database satisfies C_2 but not C_1; thus, a query optimizer that restricts its search to strategies that do not use Cartesian products may fail to find a τ-optimum strategy, if C_1 is violated. □

For any connected database of three or four relations, one can show that C_1 alone suffices to ensure that there is a τ-optimum strategy that does not use Cartesian products. We believe that this is not so for larger databases, that is, C_2 is necessary in Theorem 2, because Example 1 (Section 3) shows that C_1 alone cannot ensure there is a τ-optimum strategy that avoids Cartesian products in the case of unconnected databases. However, a combinatorial explosion makes it very difficult to construct a counterexample to prove this point.

Example 5 (Theorem 3). Consider a database with four relations specifying the majors (M) that students (S) are in, the courses (C) they take, the instructors (I) of courses, and the departments (D) instructors are in. Suppose we need to join these four relations ("How is each department serving the needs of various majors?"), and the database state is:

Major	Student	Student	Course	Course	Instructor	Instructor	Department
Math	Mokhtar	Mokhtar	Phy311	Phy311	Newton	Newton	Phy
Phy	Lin	Mokhtar	Math200	Math200	Newton	Lorentz	Math
Phy	Katina	Lin	Phy311	Math5	Lorentz	Turing	Math
		Sundram	Math5	Math200	Lorentz		
		Sundram	Art10	Phy411	Einstein		
				Math200	Einstein		

Note that this database violates C_3 (e.g., $\tau(CI \bowtie ID) > \tau(ID)$). There is only one τ-optimum strategy, namely $(MS \bowtie SC) \bowtie (CI \bowtie ID)$, which is not linear, although it does not use Cartesian products. Thus, a query optimizer that considers only linear strategies that do not use Cartesian products may not find a τ-optimum strategy, if C_3 is not satisfied.

One can verify that the database satisfies C_1 and C_2. This shows that C_1 and C_2 do not imply C_3, and that the condition C_3 in Theorem 3 cannot be relaxed to C_2, nor even to C_1 and C_2. $\square$

If the conditions for the three theorems seem restrictive, then it follows from their necessity, as demonstrated by the examples, that the assumptions underlying current query optimizers are correspondingly restrictive.

5. *Discussion*

In this section, we discuss how this work is related to database cyclicity, lossless joins, and set operations. In the process, we illustrate how our framework for studying strategies can be adapted in different situations, and list some open problems. We assume here that the reader is familiar with relational theory.

A strategy S is *monotone decreasing* if and only if for every step $[\mathbf{D}_1, R_{D_1}] \bowtie [\mathbf{D}_2, R_{D_2}]$ of S, $\tau(R_{D_1} \bowtie R_{D_2}) \leq \tau(R_{D_1})$ and $\tau(R_{D_1} \bowtie R_{D_2}) \leq \tau(R_{D_1})$. Searching for an optimal strategy from among monotone decreasing strategies is a reasonable heuristic, since such a strategy reduces the size of intermediate results at every step.

Clearly, a necessary condition for a monotone decreasing strategy to exist at all is that the final result must be smaller than every relation state; this condition is not restrictive, since it should usually be the case in practice. By Theorem 3, if a database $\mathscr{D}$ satisfies $C_3(\mathscr{D})$, then there is a linear τ-optimal strategy for $\mathscr{D}$ that is monotone decreasing. Are there more general, or different, conditions under which there would always be a τ-optimal monotone decreasing strategy?

Analogously, we define a strategy S to be *monotone increasing* if and only if for every step $[\mathbf{D}_1, R_{D_1}] \bowtie [\mathbf{D}_2, R_{D_2}]$ of S, $\tau(R_{D_1} \bowtie R_{D_2}) \geq \tau(R_{D_1})$, and $\tau(R_{D_1} \bowtie R_{D_2}) \geq \tau(R_{D_1})$. Monotone increasing strategies are of interest because any strategy that does not generate spurious tuples (i.e., every intermediate tuple appears in the final result) would be monotone increasing. In view of the relationship between C_3 and monotone decreasing strategies, we now consider a new condition.

$C_4(\mathscr{D})$: Let $\mathscr{D} = (\mathbf{D}, D)$. For all disjoint subsets $\mathbf{E}_1$ and $\mathbf{E}_2$ of $\mathbf{D}$, if $\mathbf{E}_1$ is connected, $\mathbf{E}_2$ is connected, and $\mathbf{E}_1$ is linked to $\mathbf{E}_2$, then $\tau(R_{E_1} \bowtie R_{E_2}) \geq \tau(R_{E_1})$ and $\tau(R_{E_1} \bowtie R_{E_2}) \geq \tau(R_{E_2})$.

We first verify that C_4 is satisfiable under some semantic constraint. A database $\mathscr{D} = (\mathbf{D}, D)$ is *γ-acyclic* if and only if $\mathbf{D}$ is γ-acyclic, as defined by Fagin [7]. (Recently, there is renewed interest in γ-acyclicity because of its relationship to nested transactions [13].) Two relations $(\mathbf{R}, R)$ and $(\mathbf{R}', R')$ are *consistent* [2] if and only if $R[\mathbf{R} \cap \mathbf{R}'] = R'[\mathbf{R} \cap \mathbf{R}']$, and $\mathscr{D}$ is *pairwise consistent*, or *semijoin reduced* [8], if and only if every pair of relations in $\mathscr{D}$ are consistent.

Consider now a γ-acyclic, pairwise consistent database $\mathscr{D}$, and $\mathbf{E}_1$ and $\mathbf{E}_2$ as in $C_4(\mathscr{D})$. Since $\mathbf{E}_1 \cup \mathbf{E}_2$ is connected, $(\mathbf{R}_{E_1}, R_{E_1})$ and $(\mathbf{R}_{E_2}, R_{E_2})$ must be consistent [7], and therefore $\tau(R_{E_1} \bowtie R_{E_2}) \geq \tau(R_{E_1})$ and $\tau(R_{E_1} \bowtie R_{E_2}) \geq \tau(R_{E_2})$. Thus, every γ-acyclic, pairwise consistent database satisfies C_4. What can one say about τ-optimality under this condition? For instance, does $C_4(\mathscr{D})$ imply there is a τ-optimal monotone increasing strategy for $\mathscr{D}$?

As before, a necessary condition for a monotone increasing strategy to exist is that the final result R_D must be larger than every R in D. This is not a restrictive condition for γ-acyclic databases, since there is a polynomial algorithm to semijoin reduce such databases [3], and R_D for a γ-acyclic, pairwise consistent database must contain every tuple in every relation state [8].

The condition C_4 is satisfiable under a more general form of acyclicity, but this requires a modification in the meaning of connectedness.

A database $\mathscr{D} = (\mathbf{D}, D)$ is α-acyclic if and only if $\mathbf{D}$ is α-acyclic [7]. Every α-acyclic database can be represented by a *join tree* [2], or *qual tree* [8], whose nodes form $\mathbf{D}$ and whose edges connect all relations schemes containing any given attribute. Now define a nonempty subset $\mathbf{E}$ of $\mathbf{D}$ to be *connected* if and only if there is a join tree for $\mathbf{D}$ in which $\mathbf{E}$ is connected (i.e., $\mathbf{E}$ induces a subtree), and subset $\mathbf{E}_1$ to be *linked* to $\mathbf{E}_2$ if and only if $\mathbf{F}_1 \cup \mathbf{F}_2$ is connected for some $\mathbf{F}_1 \subseteq \mathbf{E}_1$ and $\mathbf{F}_2 \subseteq \mathbf{E}_2$. Note that $\mathbf{E}_1$ and $\mathbf{E}_2$ may have a common attribute even if they are not linked to each other.

Now consider an α-acyclic, pairwise consistent database $\mathscr{D}$. (Bernstein and Chiu's reduction algorithm works for α-acyclic databases in general.) Yannakakis has shown that, for an α-acyclic $\mathbf{D}$, the join dependency $\bowtie \mathbf{D}$ implies every connected (under the new definition) subset is a lossless join [26]; moreover, Goodman and Shmueli have shown that pairwise consistency implies that $R_D[\mathbf{R}] = R$, for all relations $(\mathbf{R}, R)$ [8]. These, and the fact that the final result R_D satisfies the join dependency $\bowtie \mathbf{D}$, imply that $\mathscr{D}$ satisfies $C_4(\mathbf{D})$.

What, in the context of α-acyclic databases, can be said about τ-optimality under C_4? For instance, Yannakakis has a linear strategy for $\mathscr{D}$—whose every step is a lossless join—that has polynomial running time with respect to D and R_D, but it is not known if the strategy is τ-optimal.

Osborn has also proposed a strategy that uses only lossless joins [15]. Assuming the database $\mathscr{D} = (\mathbf{D}, D)$ satisfies a set F of functional dependencies and (1) F is embedded in $\mathbf{D}$, (2) for each $\mathbf{X} \to \mathbf{Y}$ in F, $\mathbf{X}$ is a superkey of some $\mathbf{R}$ in $\mathbf{D}$, and (3) some $\mathbf{R}$ in $\mathbf{D}$ is a superkey for $\bigcup \mathbf{D}$, then there is a strategy for $\mathscr{D}$ such that in each step $[\mathbf{E}_1, R_{E_1}] \bowtie [\mathbf{E}_2, R_{E_2}]$, $\mathbf{R}_{E_1} \cap \mathbf{R}_{E_2}$ is a superkey of either $\mathbf{R}_{E_1}$ or $\mathbf{R}_{E_2}$.

Honeyman has generalized Osborn's idea to extension joins [10], in which $\mathbf{R}_{E_1} \cap \mathbf{R}_{E_2}$ is a superkey of some $\mathbf{Y}$, where $\mathbf{Y} \subseteq \mathbf{R}_{E_1} - \mathbf{R}_{E_2}$ or $\mathbf{Y} \subseteq \mathbf{R}_{E_2} - \mathbf{R}_{E_1}$, and the join is $R_{E_1}[\mathbf{X}] \bowtie R_{E_2}$ or $R_{E_2}[\mathbf{X}] \bowtie R_{E_1}$, respectively, where $\mathbf{X} = (\mathbf{R}_{E_1} \cap \mathbf{R}_{E_2}) \cup \mathbf{Y}$. He gave an algorithm to determine, given a set of functional dependencies, a strategy (if it exists) in which every step is an extension join.

Whereas Honeyman was motivated by complexity issues when he defined extension joins, Sagiv [19] used those joins to address some semantic issues: he argued that the answer to a query, under the representative instance assumption, should be the union of certain relations, each defined by a sequence of extension joins. (Extension joins are lossless.)

Are these strategies τ-optimal? In general, if we define a *lossless* strategy to be one whose every step is a lossless join, then under what conditions would a lossless strategy be τ-optimal?

Condition C_2 may provide a starting point to answering this question. Section 4 already shows that if the database satisfies a set of functional dependencies that imply C_2, then there is a lossless strategy that is τ-optimum. Now observe that in each step $[\mathbf{E}_1, R_{E_1}] \bowtie [\mathbf{E}_2, R_{E_2}]$ of Osborn's strategy, we also have $\tau(R_{E_1} \bowtie R_{E_2}) \le \tau(R_{E_1})$ or $\tau(R_{E_1} \bowtie R_{E_2}) \le \tau(R_{E_2})$. We have seen how connectedness can be redefined to suit α-acyclicity; is there another definition that can capture the semantics of extension joins?

Other than redefining connectedness, we can also redefine joins. Consider the problem of taking the union of several relations, as in Sagiv's work, where the optimization centers on duplicate elimination. Here, the database scheme is a multiset of identical relation schemes, every two of which are therefore connected. If we define $\bowtie$ to be $\cup$, then C_4 is satisfied. What can one say about τ-optimal strategies for taking the union of relations?

In the case of intersections, C_3 and Theorem 3 are directly applicable. Again, consider the relation schemes to be completely connected, and define $\bowtie$ to be $\cap$. Then C_3 is satisfied, so by Theorem 3, there is a τ-optimal linear strategy. In other words, to minimize the number of elements generated in computing the intersection of sets $X_1, \ldots, X_n$ (where $\bigcap_{k=1}^{n} X_k \ne \phi$), it suffices to consider an evaluation of the form $(\cdots((X_{\theta(1)} \cap X_{\theta(2)}) \cap X_{\theta(3)}) \cap \cdots) \cap X_{\theta(n)}$, where θ is a permutation of $1, \ldots, n$.

ACKNOWLEDGMENT. Long, long ago, in a galaxy far, far away, Nat Goodman gave me the idea that led to these results. Many thanks to the referees, whose helpful comments led to a clarification, the introduction of the examples in Section 4, and the inclusion of Section 5.

REFERENCES

1. AHO, A. V., BEERI, C., AND ULLMAN, J. D. The theory of joins in relational databases. *ACM Trans. Datab. Syst. 4*, 3 (Sept. 1979), 297–314.
2. BEERI, C., FAGIN, R., MAIER, D., AND YANNAKAKIS, M. On the desirability of acyclic database schemes. *J. ACM 30*, 3 (July 1983), 479–513.
3. BERNSTEIN, P. A., AND CHIU, D.-M. W. Using semi-joins to solve relational queries. *J. ACM 28*, 1 (Jan. 1981), 25–40.
4. CHRISTODOULAKIS, S. Implications of certain assumptions in database performance evaluation. *ACM Trans. Datab. Syst. 9*, 2 (June 1984), 163–186.
5. CHRISTODOULAKIS, S., AND FORD, D. A. Retrieval performance versus disc space utilization on WORM optical discs. In *Proceedings of the 1989 ACM-SIGMOD International Conference on Management of Data* (Portland, Or., June). ACM, New York, 1989, pp. 306–314.
6. DEWITT, D. J., KATZ, R. H., OLKEN, F., SHAPIRO, L. D., STONEBRAKER, M. R., AND WOOD, D. Implementation techniques for main memory database systems. In *Proceedings of the 1984 ACM-SIGMOD International Conference on Management of Data* (Boston, Mass., June 18–21). ACM, New York, 1984, pp. 1–8.
7. FAGIN, R. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM 30*, 3 (July 1983), 514–550.

8. GOODMAN, N., AND SHMUELI, O. Tree queries. A simple class of relational queries. *ACM Trans. Datab. Syst. 7*, 4 (Dec. 1982), 653–677.
9. GRAEFE, G. Rule-based query optimization in extensible database systems. Ph.D dissertation, Department of Computer Science, Univ of Wisconsin-Madison, Madison, Wisc., Nov. 1987.
10. HONEYMAN, P. Extension joins. In *Proceedings of the International Conference on Very Large Data Bases* (Montreal, Canada, Oct). ACM, New York, 1980, pp. 239–244
11. IBARAKI, T., AND KAMEDA, T. On the optimal nesting order for computing N-relational joins. *ACM Trans Database Syst. 9*, 3 (Sept. 1984), 483–502.
12. KRISHNAMURTHY, R., BORAL, H., AND ZANIOLO, C. Optimization of nonrecursive queries. In *Proceedings of the International Conference on Very Large Data Bases* (Kyoto, Japan, Aug.), 1986, pp. 128–137.
13. LEVENE, M., AND LOIZOU, G. γ-acyclic database schemes and nested transactions. In *Nested Relations and Complex Objects in Databases*, S. Abiteboul, P. C. Fischer, and H.-J. Schek, eds., Springer-Verlag, Berlin, 1989, pp. 313–323.
14. ONO, K., AND LOHMAN, G. M. Measuring the complexity of join enumeration in query optimization. In *Proceedings of the International Conference on Very Large Data Bases* (Brisbane, Australia, Aug.). 1990, pp. 314–325.
15. OSBORN, S. L. Normal forms for relational databases Res. Rep. CS-78-06, Dept. of Computer Science, Univ Waterloo, Waterloo, Ont., Canada, 1978.
16. RICHARDSON, J. P., LU, H., AND MIKKILINENI, K. Design and evaluation of parallel pipelined join algorithms. In *Proceedings of the 1987 ACM-SIGMOD International Conference on Management of Data* (San Francisco, Calif., May 27–29) ACM, New York, 1987, pp. 399–409.
17. RISSANEN, J Independent components of relations. *ACM Trans. Datab Syst 2*, 4 (Dec. 1977), 317–325.
18. ROSENTHAL, A., DAYAL, U., AND REINER, D. Speeding a query optimizer: The pilot pass approach. Manuscript, 1990.
19. SAGIV, Y. Can we use the universal instance assumption without using nulls? In *Proceedings of the 1981 ACM-SIGMOD International Conference on Management of Data* (Ann Arbor, Mich., Apr. 29–May 1). ACM, New York, 1981, pp. 108–120.
20. SELINGER, P. G., ASTRAHAN, M. M., CHAMBERLIN, D. D., LORIE, P. A., AND PRICE, T. G. 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, 1979, pp. 23–34.
21. SWAMI, A. 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., June). ACM, New York, 1989, pp. 367–376.
22. SWAMI, A., AND GUPTA, A. Optimizing 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.
23. ULLMAN, J. D. *Principles of Database and Knowledge-Base Systems*, vol. 1. Computer Science Press, Rockville, Md., 1988.
24. WHANG, K. Y. Query optimization in Office-by-Example. IBM Res. Rep. RC11571. IBM T J. Watson Research Center, Yorktown Heights, N.Y. 1985.
25. WONG, E , AND YOUSSEFI, K. Decomposition—A strategy for query processing. *ACM Trans. Datab. Syst. 1*, 3 (Sept. 1976), 223–241.
26. YANNAKAKIS, M. Algorithms for acyclic database schemes. In *Proceedings of the International Conference on Very Large Data Bases* (Cannes, France, Sept.). 1982, pp. 82–94
27. YAO, S. B. Optimization of query evaluation algorithms. *ACM Trans Datab. Syst. 4*, 2 (June 1979), 133–155.

RECEIVED MARCH 1990; REVISED NOVEMBER 1990; ACCEPTED NOVEMBER 1991