Distributed Query Processing

C. T. YU AND C. C. CHANG

Department of Electrical Engineering and Computer Science, University of Illinois at Chicago, Chicago, Illinois 60680

In this paper, various techniques for optimizing queries in distributed databases are presented. Although no attempt is made to cover all proposed algorithms on this topic, quite a few ideas extracted from existing algorithms are outlined. It is hoped that large-scale experiments will be conducted to verify the usefulness of these ideas and that they will be integrated to construct a powerful algorithm for distributed query processing.

Categories and Subject Descriptors: C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*distributed databases*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; H.2.4 [**Database Management**]: Systems—*distributed systems*; *query processing*

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Communication, cyclic queries, distributed query processing, fragment processing, heuristics, join, optimization, performance, semijoin, tree queries

INTRODUCTION

Many algorithms to process queries in different distributed database systems have been proposed and implemented. In this paper, we restrict our attention to those strategies on relational databases [Codd 1970, 1972; Date 1977; Ullman 1980]. In spite of the restriction, there are numerous algorithms on the subject [Apers et al. 1983; Baldissera et al. 1979; Bernstein and Chiu 1981; Bernstein et al. 1981; Black and Luk 1982; Chang 1982a, 1982b; Cheung 1981; Chiu 1980; Chiu and Ho 1980; Epstein et al. 1978; Goodman et al. 1979; Gouda and Dayal 1981; Hevner 1980; Hevner and Yao 1979; Jarke and Koch 1983; Jarke and Schmit 1982; Kambayashi et al. 1982; Kerchberg et al. 1980; Kim 1982; Reiner 1982; Williams et al. 1981; Wong 1977, 1981; Yu and Ozsoyoglu 1979; Yu et al. 1982a, 1983, 1984a, 1984b], but they are not designed for the same environment. For example, the algorithm in Gouda and Dayal [1981] is suitable for a local network, the algorithm in Kerchberg et al. [1980] is designed for a star network, and most of the other algorithms are designed for long haul networks. Also, some environments have no fragmented relations, whereas in others some relations may be fragmented. In some situations, a query is embedded in a program and is likely to be executed repeatedly and therefore requires an extremely efficient strategy to process the query, even if the compilation cost is high. In other situations, the queries are submitted by users on an ad hoc basis, and thus a reasonably efficient strategy produced by a fast algorithm is needed. Because of this diversity, it is unlikely that a particular algorithm is suitable for all environments. In fact, no

Author's present address: C. C. Chang, Department of Computer Science, Iowa State University, Ames, Iowa 50010.

CONTENTS

large-scale experiments have been performed to demonstrate the superiority of one algorithm over all other algorithms in a given environment.

Our intention is to present some of the important ideas that have been proposed for processing queries in distributed relational systems. The ideas involve the following: the necessity for and the assumptions used in estimating the sizes of temporary relations that are created during the processing of a distributed query; the use of semijoins to reduce intersite communication cost; the separation of an algorithm based on semijoins into three phases—the copy identification phase, the reduction phase, and the assembly phase; the characterization of queries solvable by semijoins; the transformation of cyclic queries into tree queries; the optimal processing of certain restricted types of queries, enhancements of semijoin strategies in the reduction phase, and the identification of relations that need not participate in joins in the assembly phase; and the handling of fragments. These ideas are explored in the sections below.

No attempt is made to cover all proposed algorithms. Brief descriptions of some query-processing algorithms can be found in Reiner [1982]. Other issues in distributed databases can be found in Adiba et al. [1977], Rothnie and Goodman [1977a, 1977b], and Rothnie et al. [1980].

1. OPERATIONS AND COST MEASURES

In this paper a relational database [Codd 1970, 1972; Date 1977; Ullman 1980] with relations distributed in different sites is assumed. A relation is a two-dimensional table and is denoted by $R[X]$, where X is the *schema* of relation R and represents the names of columns. The relational data manipulation operations used in this paper are projection, selection, join, and semijoin [Bernstein and Chiu 1981]. They are described as follows:

Projection. The projection of relation R on a set of attributes T is denoted by $R.T$ or $R(T)$, where R is a relation with schema X, and T is a subset of X. It is obtained by discarding all columns of R that are not in T, and eliminating duplicated rows, if necessary.

Selection. The selection of those tuples whose A-attribute values equal to a specified constant in relation R is denoted by ($R.A$ = the specified constant), where A is an element of X.

It is obtained by choosing all rows of R whose A-attribute values are equal to the specified constant. One or more select clauses on the same relation may be used in selection. Operators other than "=" (e.g., $\geq$ and $\neq$) are allowed.

Join. The join of relation R_1 with relation R_2 on attribute A is denoted by ($R_1.A = R_2.A$), where R_1 and R_2 are the *joining relations*. Let X and Y be the schema of R_1 and R_2, respectively. The attribute A, which is an element of X and Y, is the *joining attribute* of R_1 and R_2.

The join is obtained by concatenating each row of R_1 with each row of R_2 whenever the A-attribute values of the two rows

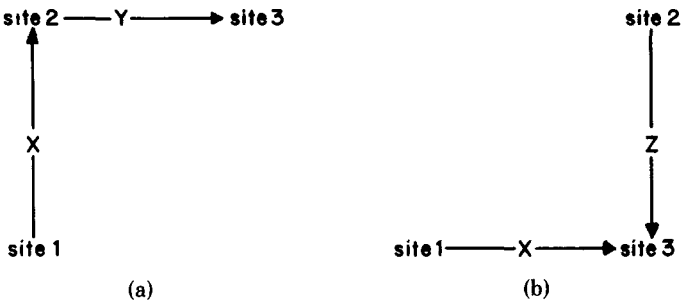


Figure 1. To answer a query, (a) X units of data are transferred from site 1 to site 2 and Y units of data are transferred from site 2 to site 3. (b) X units of data from site 1 and Z units of data from site 2 are transferred in parallel to site 3.

are equal. Since the *equality* operation results in two identical columns, one column may be eliminated. One commonly used join operation is the *natural join*, where two rows from the joining relations are concatenated whenever the corresponding values under all common attributes of the two relations are equal. We use $(R_1 = R_2)$ to denote the natural join of relations R_1 and R_2.

Semijoin [Bernstein and Chiu 1981; Bernstein and Goodman 1979; Yu and Ozsoyoglu 1979]. The semijoin from relation R_2 to relation R_1 on attribute A is denoted by $R_2 - A \rightarrow R_1$, where R_2 is the *sending relation*, R_1 is the *reduced relation*, and A is the joining attribute. Sometimes we use $R_2 \rightarrow R_1$ to represent $R_2 - A \rightarrow R_1$ if there is no need to identify the attribute. It can be obtained by joining R_1 and R_2 on attribute A, then projecting the resulting relation on the schema of R_1. Semijoins are also useful in database machines (see, e.g., Babb [1979]).

If no relation is fragmented, then in the performance of projections and selections, a local processing cost only is involved. However, when joins and semijoins are executed, communication costs between different sites may be incurred in addition to the local processing cost. For example, if R_1 and R_2 are in different sites, R_1 must be sent to the site containing R_2, or R_2 must be sent to the site of R_1 before the operation can take place.

Local processing costs usually are evaluated in terms of the number of disk accesses and CPU processing time, while communication costs are expressed in terms of the total amount of data transmitted. For geographically dispersed computer networks, communication cost is normally the dominant consideration, but local processing cost is of greater significance for local networks. In this paper, we are mostly concerned with geographically dispersed computer networks.

We assume that the cost of transferring an amount of data, say X, from one site to another site is $c_0 + c_1 * X$, where c_0 is the start-up cost of initiating transmission and c_1 is a proportionality constant. The cost for answering a query can be expressed in terms of the *total cost* measure or the *response time* measure. The total cost measure [Hevner and Yao 1979] is the sum of the costs of transferring data. In Figure 1a, where X units of data necessary to answer a query is transferred from site 1 to site 2 and Y units of data from site 2 to site 3, the total cost is $(c_0 + c_1 * X) + (c_0 + c_1 * Y) = 2c_0 + c_1(X + Y)$. The response time measure [Hevner and Yao 1979] is the time from the initiation of the query to the time when the answer is produced. In Figure 1b, where X units of data from site 1 and Z units of data from site 2 are transferred in parallel to site 3 to answer the query, the response time cost is the maximum of $c_0 + c_1 * X$ and $c_0 + c_1 * Z$. In this paper, we are mostly concerned with the total cost measure only.

Since the amount of data transferred affects the cost of a strategy, attempts have

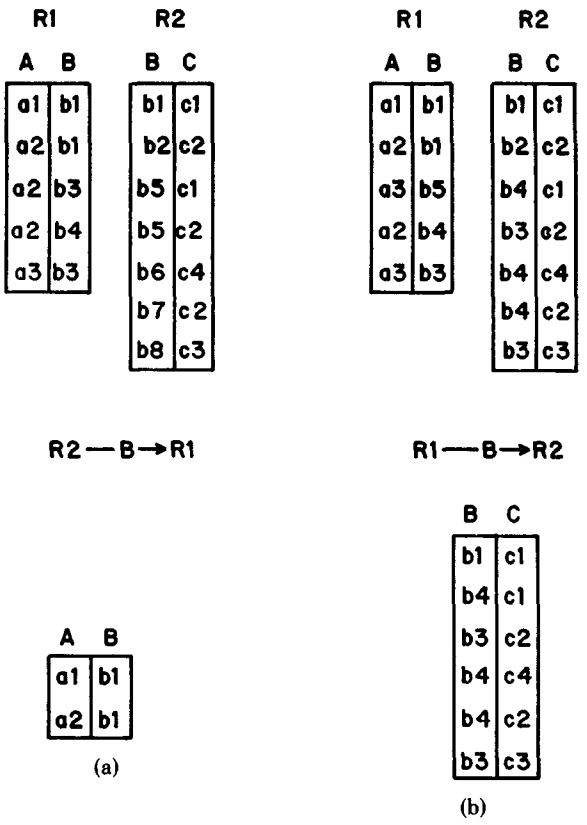


Figure 2. Illustrating semijoins.

been proposed to reduce it. A promising approach is to make use of *semijoins* [Bernstein and Chiu 1981; Hevner and Yao 1979]. For example, the *semijoin* from relation $R_2[B, C]$ to relation $R_1[A, B]$ on attribute B ($R_2 — B \rightarrow R_1$) can be obtained by projecting R_2 on attribute B, then joining the result of the projection with R_1, rather than computing the join and then the projection. We can easily see that $R_2 — B \rightarrow R_1$ is never bigger than R_1, and is usually much smaller in size. In Figure 2a, the attribute values $\{b_3, b_4\}$ of R_1 do not appear in R_2. Thus the corresponding tuples $\{(a_2, b_3), (a_2, b_4), (a_3, b_3)\}$ are eliminated from R_1. The semijoin from R_2 to R_1 consists of the retained tuples $\{(a_1, b_1), (a_2, b_1)\}$.

Suppose that R_1 and R_2 as given in Figure 2 are at different computer sites, and the join of R_1 and R_2 is desired at the site containing R_1. Suppose that each value in each of the attributes A, B, and C has unit width. To obtain a join of R_1 and R_2 at the site containing R_1, one method is that we send R_2 to R_1, then take the join at the site containing R_1. This method has a communication cost of $c_0 + c_1[7(1 + 1)] = c_0 + 14c_1$. The second method consists of sending the B-attribute values of R_1, that is, $\{b_1, b_3, b_4\}$ to the site containing R_2. Then all those tuples of R_2 whose B-attribute values do not appear in $\{b_1, b_3, b_4\}$ are eliminated; that is, $R_1 — B \rightarrow R_2$ is computed. This operation yields $\{(b_1, c_1)\}$. The reduced R_2 is then sent back to the site containing R_1 to join with R_1. In this example, the sending of R_1 projected on B costs $c_0 + c_1(3)$. The sending of the reduced R_2 costs $c_0 + c_1(1 + 1)$. Thus the second method is better than the first if $2c_0 + 5c_1 < c_0 + 14c_1$. This use of semijoin is justified in situations of small c_0. On the other hand, if the number of B-attribute values in com-

mon between R_1 and R_2 is large, as in Figure 2b, the use of semijoin may not be profitable. Clearly, it is desirable to estimate the number of attribute values in common between two relations before deciding to execute certain semijoins.

2. ESTIMATION

It can be inferred from the previous section that the performance of a distributed query-processing algorithm depends to a significant extent on the estimation algorithm used to evaluate the expected sizes of some intermediate relations. The choice of a reasonable estimation algorithm is therefore extremely important, as is described below.

Suppose that relations R_1 and R_2 are single-attribute relations, and, further, that the values of the common attribute, say A, are uniformly and independently distributed on the relations. The desired estimation is the size of $R_2 - A \rightarrow R_1$, that is, $|R_2 - A \rightarrow R_1| * w$, where $|X|$ denotes the cardinality of X and w is the average width of a tuple in R_1.

Letting p_{ia} be the probability that a value in attribute A appears in R_i, $i = 1, 2$, then p_{ia} is called the *selectivity* of R_i on attribute A. Since the values in the two relations are independently distributed, the probability that a value appears in both relations is $p_{1a} * p_{2a}$. Thus the expected number of distinct values in common between the two relations is $|A| * p_{1a} * p_{2a}$, where $|A|$ is the cardinality of the domain of the attribute A. The size of the reduced R_1 can be estimated to be $|A| * p_{1a} * p_{2a} * w$. This can be rewritten as $|R_1| * p_{2a} * w$.

In a different scenario, R_2 is the same as above but R_1 is a relation with two attributes A and B. After the semijoin, $R_2 - A \rightarrow R_1$, the cardinality of R_1 can be estimated as $|R_1| * p_{2a}$, where $|R_1|$ is the number of tuples of R_1 before the semijoin was performed. The estimation problem of the cardinality of R_1 projected on the B-attribute after the semijoin can be demonstrated in the following *ball-color* problem: "There are n balls with m different colors. Find the expected number of colors if t balls are randomly selected from the n balls." The correspondences are as follows: n balls are the number of tuples of R_1 before the semijoin, m colors are the number of distinct values of R_1 projected on the B-attribute before the semijoin, and the t selected balls correspond to the number of tuples of R_1 after the semijoin. The expected number of colors of the t selected balls is

$$g(m, n, t) = m * \left[1 - \prod_{i=1}^{t} \left(\frac{n((m-1)/m) - i + 1}{n - i + 1}\right)\right].$$

This solution is the same as the solution given by Yao [1977] in the block-access problem. It should be pointed out that, although t is a parameter given in the ball–color problem, the number of tuples of R_1 after the semijoin needs to be estimated. Some inaccuracy in the estimation can be expected. The formula, if evaluated in the present form, is computationally expensive and may cause overflow or underflow for large values of t. The following function given in Goodman et al. [1979] and Bernstein et al. [1981] is an approximation to the formula described above:

$$\begin{cases} m & \text{if } t \geq 2m, \\ \dfrac{(t+m)}{3} & \text{if } 2m \geq t \geq \left(\dfrac{m}{2}\right), \\ t & \text{if } \left(\dfrac{m}{2}\right) > t. \end{cases}$$

A semijoin strategy can be viewed as a directed graph, where the vertices are the relations and a directed edge from R_i to R_j; that is, $R_i \rightarrow R_j$ denotes the semijoin from R_i to R_j. The semijoins that are executed first are those involving nodes with in-degree = 0. For example, in the semijoin strategy $R_i \rightarrow R_j \rightarrow R_k$, R_i has in-degree = 0 and the semijoin $R_i \rightarrow R_j$ is executed first. After the execution of the semijoin, the reduced R_j, denoted by $R_{j'}$, is produced. The strategy becomes $R_{j'} \rightarrow R_k$. $R_{j'}$ has in-degree = 0, implying that the semijoin $R_{j'} \rightarrow R_k$ will be executed next. Clearly, directed cycles will not appear in a valid semijoin strategy; otherwise, the semijoin strategy does not terminate.

Now, suppose that R_1 and R_2 are the same as above, and R_3 is a single-attribute relation with attribute B. After R_1 is reduced by R_2 and R_3 using the semijoins $R_2 — A \rightarrow R_1 \leftarrow B — R_3$, the number of tuples of the resulting R_1 can be estimated as $p_{2a} * p_{3b} * |R_1|$, where p_{3b} is the selectivity of R_3 under attribute B. Thus the size of R_1 can be estimated as $p_{2a} * p_{3b} * |R_1| * w$, where w is the average width of a tuple in R_1. Moreover, the expected number of distinct A values and the expected number of distinct B values in R_1 can be estimated by using the block-access formula as described above.

We are given three relations R_1, R_2, and R_3, each having the attribute values A and B. In the following strategies, $R_1 — B \rightarrow R_2 — A \rightarrow R_3$ and $R_2 — A \rightarrow R_1 — B \rightarrow R_3$, R_3 is reduced by the same set of relations on the same attributes. In the first case, the number of distinct A-values of R_2 after executing the semijoin $R_1 — B \rightarrow R_2$ is estimated to be $g(p_{2a}|A|, |R_2|, |R_2|p_{1b})$. Thus the number of tuples of R_3 can be estimated to be $|R_3| * g(p_{2a}|A|, |R_2|, |R_2|p_{1b})/|A|$. In the latter case, the number of tuples of R_3 can be estimated to be $|R_3| * g(|B|p_{1b}, |R_1|, |R_1|p_{2a})/|B|$. Since the two expressions are in general not equal, the reduced relations R_3 are different in size for the two strategies after the execution of the semijoins. Thus estimating the size of a relation in a semijoin strategy necessitates recognizing the *history* of the operations. Such estimation algorithms are given by Bernstein et al. [1981], Luk and Black [1981], and Yu et al. [1983]. The above estimation techniques may be extended to apply to multiattribute semijoins.

Consider the semijoin $R_1 — AB \rightarrow R_2$, where both R_1 and R_2 contain attributes A and B and AB denotes the composite attribute A and B. Letting p_{iab} be the selectivity of R_i under AB, $i = 1$ or 2, define it as $|R_i(A, B)|/(|A| * |B|)$, where $|R_i(A, B)|$ is the number of tuples in the projection of R_i on AB. Then the number of tuples in the resulting R_2 can be evaluated as $p_{1ab}|R_2|$, and the expected number of distinct values of R_2 under A can be estimated via the ball-color problem with $n = |R_2(A, B)|$, $t =$ the size of $R_2(A, B)$ after the semijoin $= p_{iab}|R_2(A, B)|$, and $m = |R_2(A)|$.

We have mentioned that if R_i and R_j are at different sites, $R_i — A \rightarrow R_j$ can be computed by sending $R_i(A)$ from the site containing R_i to the site containing R_j. Other methods for computing the semijoin have been developed [Kambayashi 1981, 1982; Krishnamurthy and Morgan 1984; Sacco 1984; Wah and Lien 1984; Yu et al. 1982b]. For example, if $|R_i(A)| < |A| - |R_i(A)|$, it is cheaper to send the complement, $A - R_i(A)$. Another way is to send a bit vector indicating the presence or absence of the attribute values. These data compression techniques help to reduce data transfer.

3. THREE PHASES

We shall concern ourselves with strategies of semijoins in Sections 3–7 of this paper. The queries under consideration are of the form {target component | qualification component}, where the qualification component identifies the tuples of the relations satisfying the query and is of the form AND $(R_i.A_k = R_j.A_l)$; that is, it is the conjunction of equality clauses where the R's stand for the relations and the A's for the attributes. The target component specifies the attributes of certain relations to be outputted to the user and is of the form $(R_t.A_s, \ldots, R_g.A_n)$ as in the following example: $\{(R_1.A_1, R_3.A_3) | (R_1.A_1 = R_2.A_1)$ AND $(R_1.A_2 = R_3.A_2)\}$. For each tuple of R_1, each tuple of R_2, and each tuple of R_3 satisfying $(R_1.A_1 = R_2.A_1)$ AND $(R_1.A_2 = R_3.A_2)$, the tuple of R_1 and the tuple of R_3 are projected onto attributes A_1 and A_3, respectively, to be presented to the user.

If a clause of the form $(R_i.A_k =$ constant) appears in the qualification of the query, this clause can be processed at the local site containing R_i and therefore can be eliminated. If the qualification of a query is a disjunction of equality clauses, then each clause can be treated as the qualification of a subquery. After evaluating the subqueries, the results are merged to provide the answer to the original query. We shall restrict ourselves to the type of queries mentioned above.

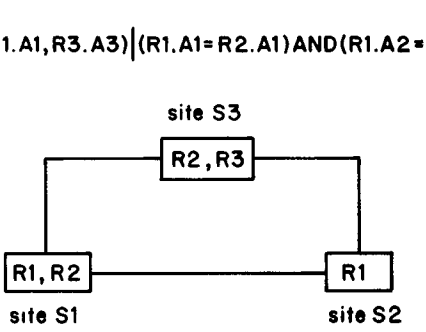


Figure 3. Three relations distributed on three sites.

The processing of distributed queries can be separated into three phases: the *copy identification phase*, the *reduction phase*, and the *assembly phase*. In the *copy identification phase*, one or more copies of every relation appearing in the qualification of the query are identified and will be used to process the query. Since a distributed database system may contain duplicate copies of some relations, the identification of appropriate copies of the relations in order to minimize the cost of transmission may not be an easy process.

The following query will serve as an example: $\{(R_1.A_1, R_3.A_3) \mid (R_1.A_1 = R_2.A_1)$ AND $(R_1.A_2 = R_3.A_2)\}$, with the three relations R_1, R_2, and R_3 distributed in the three sites S_1, S_2, and S_3, as shown in Figure 3. One way of approaching the problem would be to select the copy of R_1 from S_2, the copy of R_2 from S_1, and the copy of R_3 from S_3, but this operation would likely incur high transmission cost, because two of the three relations have to be sent to the site containing the other relation. Another method would be to have the copies of the relations R_2 and R_3 in site S_3 and the copy of the relation R_1 in S_2. In this case, since R_2 and R_3 do not have a common *joining attribute*, either R_1 is sent to site S_3 or R_2 and R_3 are sent to S_2. Still another approach would be to have copies of R_1 and R_2 at site S_1 and the copy of R_3 at site S_3. This last choice is the best of the three, because (1) R_1 and R_2 projected on A_1 can be merged together without communication cost to produce a relation that is not larger than the original relation R_1, and (2) it is then sufficient either to transfer the merged relation from S_1 to S_3 or to transfer R_3 from S_3 to S_1.

Suppose that a query references n relations. If relation R_i has X_i copies, $1 \leq i \leq n$, then a straightforward enumerative algorithm to choose one copy for each relation takes time $O(\prod_{i=1} x_i)$. This is exponential in time. It turns out [Yu et al. 1982b] that finding one copy among several possibilities of each relation referenced by a given query so that the cost of answering the query is minimized is a NP-hard problem (in the number of sites having at least one copy of a relation referenced by the query). This situation holds true even when restricted to the *simple queries* (all relations have one and exactly the same attribute) in a *fully connected network* (i.e., each site can communicate directly with every other site).

In the *reduction phase*, semijoins are usually used to eliminate tuples of the relations that do not satisfy the qualification of the query. For example, for the query cited earlier with the best choice of the copies of the relations, one could perform semijoin $R_2 - A_1 \rightarrow R_1$ to eliminate some tuples of R_1 without incurring communication cost. If the result of the semijoin were to be $R_{1'}$, other semijoins could then be performed $R_{1'} - A_2 \rightarrow R_3$ to reduce R_3, or semijoin $R_3 - A_2 \rightarrow R_{1'}$ to further reduce $R_{1'}$.

In the *assembly phase*, relations in the qualification component of the query are sent to one site to produce the output required by the user. For example, in the above query, $R_{1'}$ (which is the result of

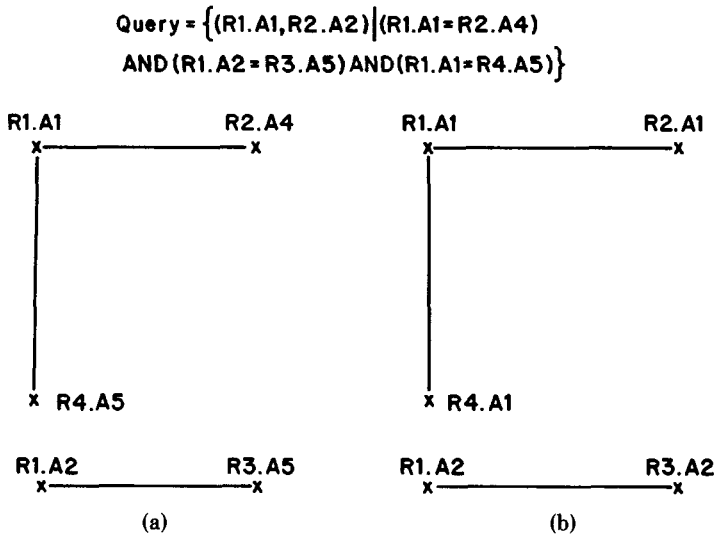


Figure 4. Representing a query by its join-graph: (a) join-graph; (b) join-graph with attributes renamed.

semijoin $R_2 - A_1 \rightarrow R_1$) can be sent to S_3 to be merged with R_3 to produce the output.

We should like to point out that the separation of the query-processing strategy into the three phases may not yield the least transmission cost; rather, it tends to simplify the concepts involved.

A reasonable strategy for choosing the copies of the relations is to find the minimum number of sites containing chosen copies of the relations. Unfortunately, that is also a NP-complete problem [Yu et al. 1982b]. However, since the number of relations referenced by a query and the number of sites containing those relations is usually small, finding the number of sites by enumeration does not require much time. The reduction phase and the assembly phase will be described in more detail in the subsequent sections.

4. TREE QUERIES VERSUS CYCLIC QUERIES

4.1 Characteristics

Only certain types of queries can be solved using semijoins. More precisely, a relation appearing in the qualification of a query is said to be *fully reduced* if all tuples not satisfying the qualification of the query have been eliminated. It is clear that if the joins of all the relations in the qualification are taken, and the resulting relation is then projected back onto the attributes of the original relations, then the projected relations will then be fully reduced, because any tuple of each projected relation not satisfying the qualification would have been eliminated by the joins. If semijoins are used to reduce relations, less communication cost may be incurred. However, depending on the type of query, the relations appearing in the query may not be fully reduced. As a result, communication cost in assemblying the relations can still be high. A precise characterization of the type of queries whose referenced relations can be fully reduced by semijoins is therefore desirable. The characterization is facilitated by defining a *join-graph* and a *query-graph* [Bernstein and Chiu 1981].

The vertices of a *join-graph* are described as $\{R_i.A_j \mid R_i$ is a relation, A_j is an attribute, and $R_i.A_j$ appears in a clause of the qualification$\}$. The edges of the graph represent the equality clauses. As shown in Figure 4a, each $R_i.A_j$ is a vertex, and an equality clause of the form $(R_i.A_j = R_k.A_1)$ is represented by an edge between $R_i.A_j$ and $R_k.A_1$. Since equality is a transitive operator, $(R_i.A_j = R_k.A_1)$ AND $(R_k.A_1 = R_i.A_m)$

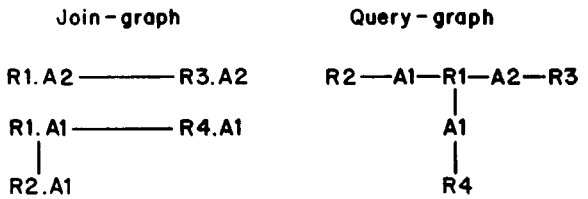


Figure 5. An example of join-graph and query-graph.

imply $(R_i.A_j = R_t.A_m)$. If two or more attributes of a relation are transitively related, it is sufficient to retain one of them, since the other can be eliminated by local processing.

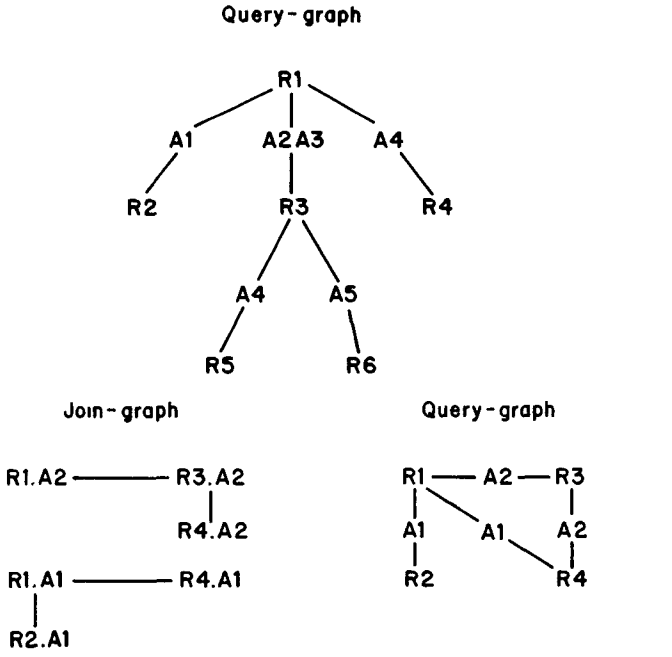
We can thus rename all attributes that are transitively related to be the same attribute. In Figure 4b, A_1, A_4, and A_5 in one component of the join-graph are all renamed to be A_1, while A_2 and A_5 in another component are renamed to be A_2. In other words, all vertices in a connected component refer to the same joining attribute, and the connected component can be uniquely identified by the attribute. If $R_i.A_k$ and $R_j.A_k$ are in the same connected component identified by attribute A_k, then clearly $R_i - A_k \rightarrow R_j$ and $R_j - A_k \rightarrow R_i$ are possible semijoins. On the other hand, $R_i.A_k \rightarrow R_j.A_t$, for $t \neq k$, is not a possible semijoin because $(R_i.A_k = R_j.A_t)$ is neither stated nor implied by the qualification. In this manner all possible semijoins of the qualification are of the form $R_i - A_k \rightarrow R_j$ or $R_j - A_k \rightarrow R_i$ for some attribute A_k, after the renaming of the attributes.

The vertices of the *query-graph* are the relations appearing in the qualification. An edge (R_i, R_j) with label A_k exists in the query-graph if $(R_i.A_k = R_j.A_k)$ is a clause in the qualification. If $(R_i.A_l = R_j.A_l)$ also appears in the join-graph, the label of the edge in the query-graph is $\{A_k, A_l\}$; that is, the label is to include all attribute names that participate in the clauses involving the relations R_i and R_j. For example, Figure 5 illustrates a join-graph and its corresponding query-graph.

If a query-graph is a tree in the graph-theoretical sense, then it can be shown [Bernstein and Chiu 1981] that a sequence of semijoins can fully reduce all the relations. The sequence suggested by Bernstein and Chiu [1981] is rather simple. A relation is chosen arbitrarily in the query-graph as the root of the tree, for example, R. Then the leaves of the tree are well defined. For example, in Figure 6 the leaves are R_2, R_5, R_6, and R_4. The process of fully reducing all relations consists of two phases: (1) "leaves to root," which fully reduces the root relation, and (2) "root to leaves," which fully reduces the other relations after carrying out Phase 1.

The "leaves to root" phase consists of taking semijoins from each relation to its parent, starting from the leaves and ending at the root. Semijoins from relations to their common parent all should be taken before any operation of the parent with its immediate ancestor is taken. For example, in Figure 6, the semijoins $R_5 - A_4 \rightarrow R_3$ and $R_6 - A_5 \rightarrow R_3$ are taken before the semijoin $R_3 - A_2A_3 \rightarrow R_1$ is executed. The clause $(R_5.A_4 = R_3.A_4)$ is satisfied intuitively by R_3 after the semijoin $R_5 - A_4 \rightarrow R_3$ is taken. Similarly, the clause $(R_3.A_5 = R_6.A_5)$ is satisfied by R_3 after the execution of the semijoin $R_6 - A_5 \rightarrow R_3$. Thus R_3 satisfies the two clauses after application of the two semijoins. Similar arguments show that at the end of the first phase, the root relation R_1 will satisfy the clauses $\{(R_5.A_4 = R_3.A_4), (R_3.A_5 = R_6.A_5), \ldots, (R_4.A_4 = R_1.A_4)\}$; that is, the relation R_1 is fully reduced.

In the second phase ("root to leaves"), the fully reduced root relation, let us say R, is used to reduce its immediate descendants. When the semijoin from R to an immediate descendant, for example, R_i, is taken, R_i is fully reduced. This can be demonstrated by constructing a tree with R_i as the root. The "leaves to root" phase with R_i as root will be executed after the completion of the "leaves to root" phase with R as



(a)

To fully reduce R1

(R5—A4→R3, R6—A5→R3,
R2—A1→R1, R3—A2A3→R1,
R4—A4→R1)

To fully reduce the other relations after R1 is fully reduced

(R1—A1→R2, R1—A2A3→R3,
R1—A4→R4, R3—A4→R5,
R3—A5→R6)

Figure 6. A sequence of semijoins fully reduces the relations.

root plus the semijoin $R \rightarrow R_i$. In Figure 6, the sequence of semijoins ($R_5 - A_4 \rightarrow R_3$, $R_6 - A_5 \rightarrow R_3$, $R_2 - A_1 \rightarrow R_1$, $R_3 - A_2A_3 \rightarrow R_1$, $R_4 - A_4 \rightarrow R_1$) (the sequence is the "leaves to root" phase with R_1 as root) followed by $R_1 - A_4 \rightarrow R_4$ contains the sequence of semijoins ($R_5 - A_4 \rightarrow R_3$, $R_6 - A_5 \rightarrow R_3$, $R_2 - A_1 \rightarrow R_1$, $R_3 - A_2A_3 \rightarrow R_1$, $R_1 - A_4 \rightarrow R_4$) (the sequence is the "leaves to root" phase with R_4 as root). Thus R_4 will be fully reduced. The process of using the newly fully reduced relation to fully reduce its immediate descendants is continued until all the leaves are reached; at this point, all relations are fully reduced. This process is illustrated in Figure 6.

This discussion should make clear that if the query-graph of the qualification of a query is a tree, the relations can be fully reduced by semijoins. Even if the query-graph of a given qualification should be cyclic, an equivalent qualification exists that uses a tree query-graph, as is demonstrated in Figure 7a. The qualification is equivalent to that given in Figure 7b because ($R_1.A_2 = R_3.A_2$) AND ($R_3.A_2 = R_4.A_2$) is equivalent to ($R_1.A_2 = R_3.A_2$) AND ($R_1.A_2 = R_4.A_2$). The latter qualification has a tree query-graph and therefore is solvable by semijoins.

The definition of a *tree query* is that the query graph of its qualification or an equivalent qualification is a tree. A query is a *cyclic query* if none of the query-graphs of equivalent qualifications is a tree. As illustrated earlier, if the query is a tree query, the relations of a tree query can be fully reduced by semijoins, but semijoins may be

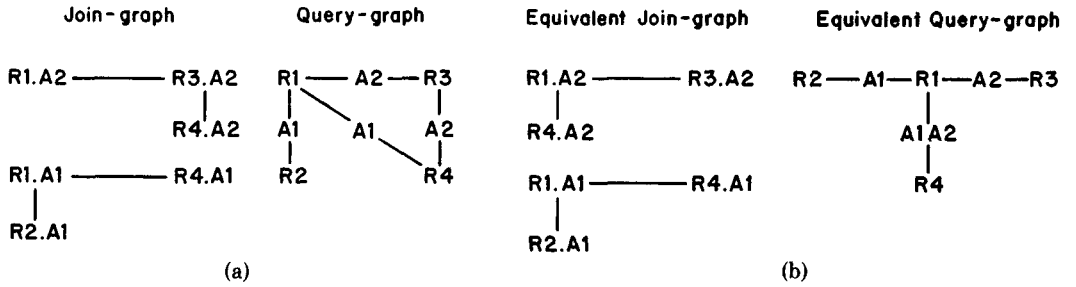


Figure 7. An example of equivalent query-graph.

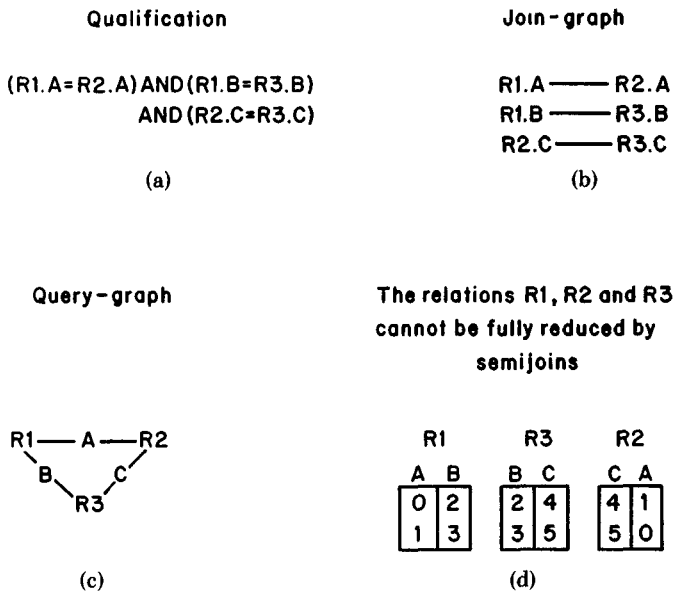


Figure 8. An illustration that the relations in a query having a cyclic query-graph cannot be fully reduced by semijoins.

inadequate to fully reduce a cyclic query [Bernstein and Chiu 1981; Bernstein and Goodman 1981]. This is illustrated by the following example. The relations R_1, R_2, and R_3 are referred to by the query as given in Figure 8d. If the semijoin $R_1 - A \rightarrow R_2$ is used, then R_2 remains unchanged because $R_1.A$ and $R_2.A$ are identical. Similarly, the semijoins $R_2 - C \rightarrow R_3$ and $R_3 - B \rightarrow R_1$ have no effect on R_3 and R_1, respectively. However, the fully reduced R_1, R_2, and R_3 should be the null relations, because no tuple of R_1, of R_2, and of R_3 simultaneously satisfies the qualification.

4.2 Tree Query Recognition Algorithm

Section 4.1 illustrates the importance of recognizing qualifications that have either tree query-graphs or are equivalent to other qualifications having tree query-graphs. It turns out that there is a simple algorithm [Graham 1979; Yu and Ozsoyoglu 1979] for the recognition of such queries, as follows. The algorithm takes a query as input, and it has two key steps. Initially, for each relation R_t, the set of attributes of the relation appearing in the qualification, $J(R_t)$, is constructed. As described earlier,

each attribute A of R_t in the qualification denotes a relationship between R_t and the set of relations containing A.

In the first step, R_i is eliminated from consideration, if a pair of relations R_i and R_j exists such that $J(R_i) \subseteq J(R_j)$. Condition $J(R_i) \subseteq J(R_j)$ guarantees that an equivalent qualification can be constructed by substituting each clause of the form $R_i.X = R_k.X$, where $k \neq j$, with two clauses $(R_i.X = R_j.X)$ AND $(R_j.X = R_k.X)$. (This substitution may produce some duplicated clauses.) After substitution, R_i appears only in the clause $R_i.X = R_j.X$. It is then clear that in the query-graph, R_i is only adjacent to R_j, and therefore is not part of any cycle. Hence the elimination of R_i will not change the type of the query-graph. In Figure 7, $J(R_3) = \{A_2\} \subseteq J(R_1)$. The edge between R_3 and R_4 is replaced by that between R_1 and R_4 (and that between R_1 and R_3). As a result, R_3 is not part of any cycle and can be eliminated without affecting the type of the query.

In the second step, if any relation is eliminated in Step 1, it is checked to determine whether it causes the elimination of an attribute. An attribute is to be eliminated if only one relation remains containing that attribute. (One should here recall that if a set of relations contains the same attribute, they are related by the equality of that attribute; thus if there is no more than one relation having that attribute, no such relationship exists.) For example, if $R_i.A = R_j.A$ is the only clause involving attribute A, and if R_i is eliminated in Step 1, then attribute A will be eliminated in this step. It is clear that the elimination of an attribute causes the updating of the relation R (more precisely, $J(R)$) having that attribute originally. The algorithm is simply an iteration of Steps 1 and 2. If all relations are eliminated at the end of the algorithm, the original query is then a tree query, because the algorithm does not affect the type of a query (tree or cyclic) and a null query is clearly a tree query. If some relations do exist at the end of the algorithm, then it can be shown that the original query is a cyclic query [Yu and Ozsoyoglu 1979]. Figure 9 illustrates the operation of the algorithm on the query given in Figure 7a.

$J(R_1) = \{A_1, A_2\}$
$J(R_2) = \{A_1\}$
$J(R_3) = \{A_2\}$
$J(R_4) = \{A_1, A_2\}$

Since $J(R_2) \subseteq J(R_1)$, eliminate R_2.
$J(R_3) \subseteq J(R_1)$, eliminate R_3.
$J(R_4) \subseteq J(R_1)$, eliminate R_4.

Since A_1 occurs in R_1 only, eliminate A_1.
A_2 occurs in R_1 only, eliminate A_2.
R_1 does not have any attribute, eliminate R_1.

All relations are eliminated. Thus, this is a tree query.

Figure 9. Demonstrating that the query in Figure 7a is a tree query.

A further characterization of cyclic queries should illustrate the concept more clearly [Goodman and Shmueli 1983]. There are two basic forms of cyclic queries as shown in Figure 10.

It is clear that the two operations used to determine a tree query will not eliminate any attribute or relation from the above query-graphs, and therefore Aring and Aclique are cyclic queries.

All other cyclic queries can be reduced to either an Aring or an Aclique by repeated applications of the two operations and by eliminating a common set of attributes from each of the relations. In essence, the absence of Aring and Aclique implies that the query is a tree query. Numerous other characterizations of cyclic and tree queries are given in Beeri et al. [1981, 1983], Fagin et al. [1980], and Goodman and Shmueli [1983] relating distributed query processing to dependency theory, database schema design, and graph theory; these processes are not covered in this paper.

4.3 Transforming a Cyclic Query into a Tree Query

Since the tree query is fully reducible, algorithms capable of transforming a cyclic query to a tree query are desirable [Goodman and Shmueli 1982b; Kambayashi and Yoshikawa 1983; Kambayashi et al. 1982].

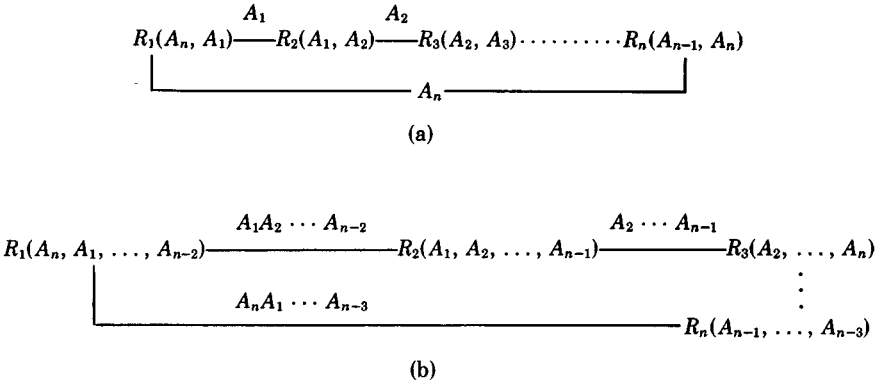


Figure 10. (a) *A* ring of *n*-relationships; (b) *A* clique of *n*-relationships.

Basically, there are three different transformation algorithms: (1) a relation-merging algorithm [Goodman and Shmueli 1982a; Kambayashi and Yoshikawa 1983], (2) a tuplewise decomposition algorithm, and (3) an attribute addition algorithm [Kambayashi et al. 1982]. Some details of the algorithms are as follows:

(1) The relation-merging algorithm simply joins certain relations residing in a cycle to eliminate the cycle. For example, given a cyclic query as shown in Figure 11a, the algorithm can join any two relations in the cycle. This causes the cycle to disappear. Figure 11b shows the query-graph resulting by joining R_2 and R_3 together.

(2) The tuple-wise decomposition algorithm is based on the tuple-substitution idea of Wong and Youssefi [1976]. The algorithm eliminates a cycle by decomposing a cyclic query into a number of tree subqueries. By first arbitrarily selecting a relation in a cycle, it constructs a tree subquery for each tuple of the relation by substituting the attribute values of the tuple. Using the query in Figure 11a, if relation R_3 is selected, $|R_3|$ subqueries will be generated, with each subquery corresponding to a tuple of R_3. Each subquery has a query graph as shown in Figure 11c. The answer of the query is then the union of all the answers of the subqueries.

(3) The attribute addition algorithm aims at fully reducing a relation in a cyclic query. In Kambayashi et al. [1982], certain attributes of some relations involved in cycles are added to other relations in such a way that a tree query results. Semijoins can be then used to fully reduce any given relation. The algorithm takes as input a cyclic query and a relation to be fully reduced, for example, R. It then chooses a spanning tree from the query-graph of the query with R as the root. Given the same example in Figure 11a, suppose that R_1 is to be fully reduced. Figure 11d–f gives the three possible spanning trees (the solid edges in each figure) with R_1 as the root. Since the query is cyclic, at least one edge is not in the spanning tree. For each edge not in the spanning tree (the dotted edge in each figure), the label of the edge is added to those of the edges that form a cycle with the edge. For example, in Figure 11d, $R_3 - B - R_1$ is the edge not in the spanning tree. Its label B is added to the labels of the edges $R_1 - A - R_2$ and $R_2 - C - R_3$ to form the new edges $R_1 - AB - R_2$ and $R_2 - BC - R_3$. (At this point, R_2, which originally does not have attribute B, is assumed to have all distinct values of B so that semijoins on attribute B involving R_2 will not produce null relations.) After this process, Figure 11d becomes a tree query. By the "leaves to root" algorithm given in Bernstein and Chiu [1981], R_1 can be fully reduced if the semijoins $R_3 - BC \rightarrow R_2$ and $R_2 - AB \rightarrow R_1$ are executed. Notice that if attribute addition had not been used, the

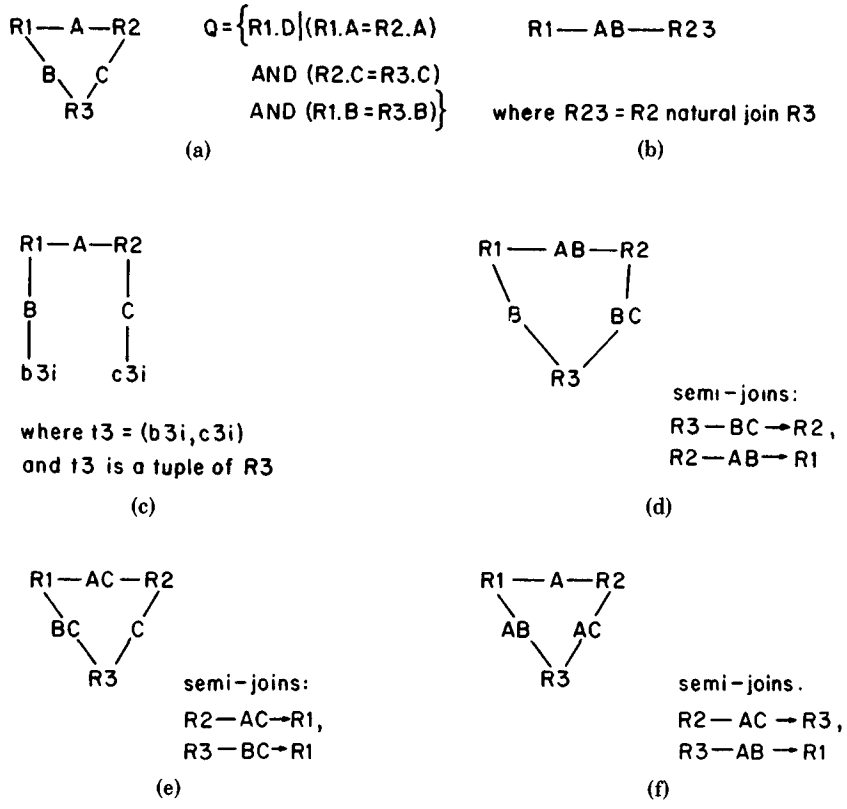


Figure 11. Transforming cyclic query into tree query.

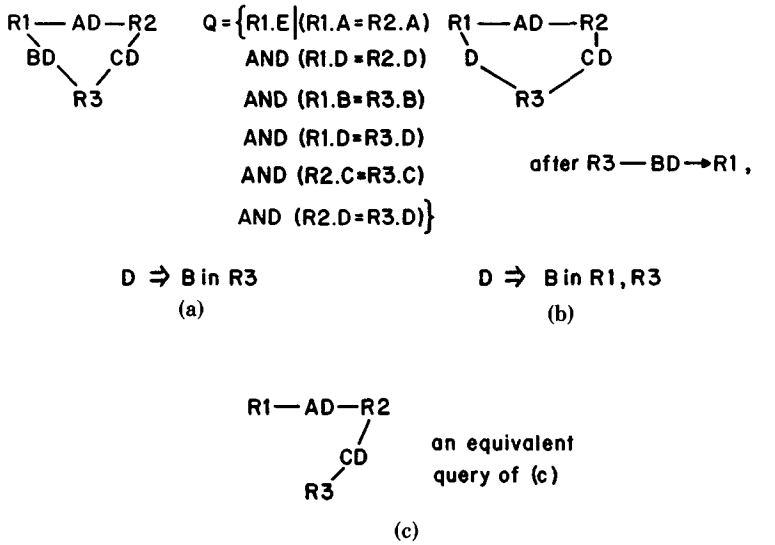


Figure 12. Functional dependency and query graph.

corresponding semijoin sequence would have been $R_3 — C \rightarrow R_2$, $R_2 — A \rightarrow R_1$, and R_1 might not have been fully reduced.

Kambayashi and Yashikawa [1983] have studied the effect of functional and multivalued dependencies on query processing, and have identified a sufficient condition for a relation in a cyclic query to be fully reduced. Given a cyclic query as shown in Figure 12a, $D \Rightarrow B$ can be considered a functional dependency identified in R_3. By executing a semijoin $R_3 — BD \rightarrow R_1$, the resulting relation R_1 will preserve the functional dependency $D \Rightarrow B$. Thus in the current database, the values of attribute B are uniquely determined if those of attribute D are known. Consequently, attribute B can be deleted, and the query-graph is changed to Figure 12b. Since all three relations contain attribute D, the query-graph is further simplified to Figure 12c, which is a tree. A relation in a cyclic query can therefore be fully reduced by semijoins if, "for each cycle in the query-graph,

(1) all relations in the cycle contain the attribute(s) X,
(2) an edge in the cycle is labeled XY, and
(3) a relation on the edge has the functional dependency $X \Rightarrow Y$."

5. OPTIMAL STRATEGY FOR SIMPLE QUERIES

In the reduction phase, the query-processing algorithms described in the previous section are all heuristics and may not always yield optimal strategies. In this section we present an optimal algorithm for *simple queries* (all relations appearing in the qualification of such a query have the same attribute, and each relation has a single attribute). The discussion will proceed under the following assumptions. Let the common attribute be A. Let the relations be $\{R_1, R_2, \ldots, R_n\}$ situated at different sites. Let R_s be the *result site* where the answer is to be produced.

As noted in Section 2, a strategy can be represented by a directed graph where the vertices are the relations and the edges represent the semijoins. It is clear that a strategy should have all paths directed toward the result site R_s and that the strategy should contain the n relations, some of which may appear more than once. The cost of a strategy is the sum of the data transmission cost of executing the semijoins represented by the edges. Since it is impossible to find the precise cost of a strategy before its execution, the usual procedure is to minimize the expected cost. As an example, if a strategy is $R_{i1} — A \rightarrow R_{i2} \cdots — A \rightarrow R_{im}$, $m = n$, $R_{im} = R_s$ and if p_{ij} is the selectivity of relation R_{ij} on attribute A, then the expected transmission cost of the strategy is $|A|[p_{i1} + p_{i1}p_{i2} + \cdots + p_{i1}p_{i2} \cdots p_{im-1}]$. An optimal strategy is one having the smallest expected cost among the directed graphs satisfying the conditions noted above.

Some properties are satisfied by an optimal strategy for a simple query. They are listed as follows:

Property 5.0. All relations should appear on one directed path; that is, an optimal strategy for a simple query is a "string" of directed edges of the form $R_{i1} \rightarrow R_{i2} \rightarrow R_{i3} \rightarrow \cdots \rightarrow R_{it}$.

In Hevner and Yao [1979], two cases are considered. In the first case, the result site is one of the sites containing $R_1, R_2, \ldots, R_n$. In the other case, the site does not contain any of the n relations.

If the result site does not contain any of the n relations, then all of the relations in an optimal strategy $R_{i1} \rightarrow R_{i2} \rightarrow \cdots \rightarrow R_{it}$ are distinct; that is, no relation appears more than once. This conclusion is rather obvious, because if a relation occurs twice or more, then the second and subsequent occurrences of the relation can be removed from the strategy, yielding an equivalent but lower cost strategy. They are equivalent because the last relation in both strategies satisfies $R_1.A = R_2.A = \cdots = R_n.A$; the latter strategy has a lower cost because the second and subsequent occurrences of the relation, having appeared earlier, will not provide extra reduction to later relations.

Property 5.1 [Hevner and Yao 1979]. All relations should appear in ascending order of size.

Thus, the optimal strategy is in fact $R_1 \rightarrow R_2 \rightarrow \cdots \rightarrow R_n \rightarrow R_s$, where $|R_1| \leq |R_2| \leq \cdots \leq |R_n|$.

One can verify the result by a straightforward check to see that if $R_{i1} \rightarrow R_{i2} \cdots \rightarrow R_{ij-1} \rightarrow R_{ij} \rightarrow R_{ij+1} \rightarrow R_{ij+2} \rightarrow \cdots \rightarrow R_{it}$ is a strategy with $|R_{ij}| > |R_{ij+1}|$, then the strategy $R_{i1} \rightarrow R_{i2} \rightarrow \cdots \rightarrow R_{ij-1} \rightarrow R_{ij+1} \rightarrow R_{ij} \rightarrow R_{ij+2} \rightarrow \cdots \rightarrow R_{it}$ yields a lower cost.

If the result site R_s contains a relation R_i, then the optimal strategy [Hevner and Yao 1979] is either $R_1 \rightarrow R_2 \rightarrow \cdots \rightarrow R_{i-1} \rightarrow R_i \rightarrow R_{i+1} \rightarrow \cdots \rightarrow R_s$ or $R_1 \rightarrow R_2 \rightarrow \cdots \rightarrow R_{i-1} \rightarrow R_{i+1} \rightarrow \cdots \rightarrow R_s$, where the former strategy passes through the result site twice and the latter strategy passes through the result site once. The lower cost strategy between the two strategies is taken as the optimal one. Optimal strategies can thus be easily obtained for simple queries.

6. OPTIMAL STRATEGY FOR TREE QUERIES

In this section, we provide an outline of a method to obtain optimal strategies to fully reduce a relation for tree queries with the restriction that any two relations have at most one single common joining attribute. The method will be illustrated by a special type of tree query in which there are m single attribute relations, each having the joining attribute A; these are labeled $A_1, A_2, \ldots, A_m$ with $|A_1| \leq |A_2| \leq \cdots \leq |A_m|$. There are n single attribute relations, each having the joining attribute B, which are labeled $B_1, B_2, \ldots, B_n$ with $|B_1| \leq |B_2| \leq \cdots \leq |B_n|$, and a two-attribute relation I having the joining attributes A and B. We are given a query referring to the above m A's, n B's, and the I relation. OPTS(m, n, I) is an optimal strategy to fully reduce some relation Y in the query, where Y should be the last relation in the sequence, and OPTS(m, n, I, X) an optimal strategy to fully reduce a specific relation X in the query.

Some properties of OPTS(m, n, I) are given as follows [Chen and Li 1983; Yu et al. 1979, 1982a]:

Property 6.0. Relations having a common attribute should all appear on one directed path. (This is the generalized version of Property 5.0 for arbitrary queries.)

Example 6.1

(a) $A_2 \rightarrow A_1 \rightarrow I \rightarrow B_2 \rightarrow B_3$, with $B_1 \rightarrow I$

does not violate Property 6.0.

(b) $A_1 \rightarrow A_2 \rightarrow I \rightarrow B_1 \rightarrow B_2 \rightarrow B_3 \rightarrow I$

is a possible optimal strategy.

(c) $A_1 \rightarrow A_2 \rightarrow I$, with $B_3 \rightarrow I$, $B_2 \rightarrow B_3$, $B_1 \rightarrow B_3$

cannot be an optimal strategy, because B_1 and B_2 are not on the same directed path. This violates Property 6.0.

(d) $A_1 \rightarrow I \rightarrow B_1 \rightarrow B_2 \rightarrow B_3 \rightarrow I \rightarrow A_2$, with $B_1 \rightarrow I$

does not violate Property 6.0.

(e) $A_1 \rightarrow I \rightarrow A_2 \rightarrow I \rightarrow B_2 \rightarrow B_3$, with $B_1 \rightarrow I$

is a possible optimal strategy.

(f) $A_1 \rightarrow I \rightarrow A_2 \rightarrow I \rightarrow B_2 \rightarrow B_3$, with $B_1 \rightarrow I$

violates Property 6.0 because the first occurrence of I and B_1 are in different paths. □

Property 6.1. All A's and B's appear exactly once, while I may appear once or more.

Example 6.2. The strategy in Example 6.1(d) violates Property 6.1 since B_1 appears twice in the strategy. □

Property 6.2. Single-attribute relations must appear in ascending order of their sizes in the path leading to the first fully reduced relation Y.

Example 6.3. The strategy in Example 6.1(a) violates Property 6.2, because A_1 and A_2 are in descending order of size; the strat-

egies in Example 6.1(b) and 6.1(e) satisfy Property 6.2. □

By Property 6.2, the first fully reduced relation in OPTS(m, n, I) is A_m or B_n or I. Thus OPTS(m, n, I) has one of the following three forms:

$$\text{OPTS}(m, n, I, A_m),$$

$$\text{OPTS}(m, n, I, B_n),$$

$$\text{OPTS}(m, n, I, I).$$

Consider OPTS(m, n, I, A_m). The vertex immediately preceding A_m cannot be a B-relation since if a B-relation is sent to A_m, the sending is a waste because it cannot merge with A_m directly. This vertex therefore must be either I or A_{m-1}, by Property 6.2.

Subcase 1. If the vertex is A_{m-1}, the set of relations preceding A_m then forms a substrategy involving the $m - 1$ A-relations $\{A_1, A_2, \ldots, A_{m-1}\}$, the n B-relations $\{B_1, B_2, \ldots, B_n\}$, and the I-relation. This substrategy denoted by OPTS($m - 1$, n, I, A_{m-1}) is optimal among all substrategies ending at A_{m-1} and involving the same subset of relations. (Otherwise, a better substrategy followed by the data transfer of the reduced A_{m-1} to A_m will produce a better strategy.)

Subcase 2. If the vertex is I, then again we have an optimal substrategy involving the same subset of relations. This substrategy is denoted by OPTS($m - 1$, n, I, I), since the last vertex in the substrategy is I.

Both substrategies process the same set of relations, and the relation immediately following each of these substrategies is A_m. The amount of data thus transmitted from each of those substrategies to A_m is identical and can be denoted by Z. OPTS(m, n, I, A_m) is either A_m preceded by OPTS($m - 1$, n, I, I) or A_m preceded by OPTS($m - 1$, n, I, A_{m-1}). If C(strategy) is the cost of the strategy, then

$$C(\text{OPTS}(m,n,I,A_m)) = (c_0 + c_1 * Z) + \min\{C(\text{OPTS}(m-1,n,I,A_{m-1})), C(\text{OPTS}(m-1,n,I,I))\}.$$

Pictorially, OPTS(m, n, I, A_m) can be represented by

$$A_m \leftarrow \min\{\text{OPTS}(m-1, n, I, I), \text{OPTS}(m-1, n, I, A_{m-1})\}, \quad (6.1)$$

where the cost functions are not explicitly written. Similarly, OPTS(m, n, I, B_n) is

$$B_n \leftarrow \min\{\text{OPTS}(m, n-1, I, I), \text{OPTS}(m, n-1, I, B_{n-1})\}. \quad (6.2)$$

If the first fully reduced relation I of OPTS(m, n, I, I) has in-degree 1, then the relation immediately preceding I can be either A_m or B_n. The two subcases are, respectively,

$$I \leftarrow A_m \leftarrow \min\{\text{OPTS}(m-1,n,I,I), \text{OPTS}(m-1,n,I,A_{m-1})\}, \quad (6.3)$$

$$I \leftarrow B_n \leftarrow \min\{\text{OPTS}(m,n-1,I,I), \text{OPTS}(m,n-1,I,B_{n-1})\}. \quad (6.4)$$

If the first fully reduced relation I has in-degree 2, then by Property 6.0 the optimal strategy is

$$\begin{array}{l} \min\{\text{OPTS}(m,0,0,A_m), \\ \quad \text{OPTS}(m,0,IA,A_m)\} \\ \swarrow \\ I \\ \nwarrow \\ \min\{\text{OPTS}(0,n,0,B_n), \\ \quad \text{OPTS}(0,n,IB,B_n)\} \end{array} \quad (6.5)$$

where IA and IB are the projections of I on the attributes A and B, respectively.

OPTS(m, n, I) is the minimal cost strategy among the five strategies given by (6.1)–(6.5) (see Example 6.4 for the end cases). It is clear from the equations above that OPTS(m, n, I, Y), where $Y = A_m$, B_n, or I, can be computed in constant time if OPTS($m - 1$, n, I, I), OPTS($m - 1$, n, I, A_{m-1}), OPTS(m, $n - 1$, I, I), OPTS(m, $n - 1$, I, B_{n-1}), OPTS(m, 0, 0, A_m), OPTS(m, 0, IA, A_m), OPTS(0, n, IB, B_n), and OPTS(0, n, 0, B_n) are known. The following method is suggested to obtain the optimal strategy.

In the two-dimensional figure in Figure 13, the point (i, j) denotes three optimal strategies involving $\{A_1, \ldots, A_i, B_1, \ldots, B_j, I\}$, one ending in A_i, one ending in B_j,

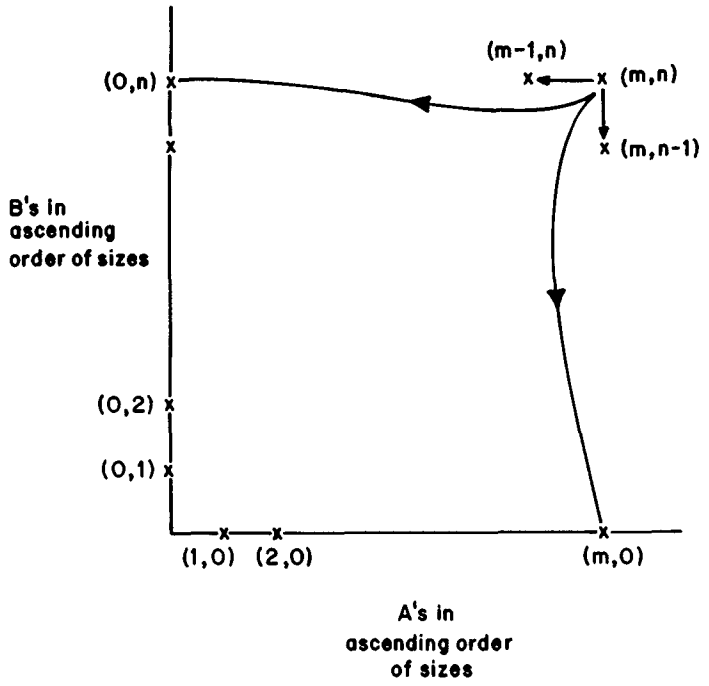


Figure 13. Illustration of how the optimal strategy is obtained.

and one ending in I. From eqs. (6.1)–(6.5), the optimal strategies at (m, n) can be obtained from those at $(m-1, n)$, $(m, n-1)$, $(m, 0)$, and $(0, n)$. If we compute all optimal strategies at (x_1, x_2), $x_1 + x_2 = t$, and at the boundary points $(i, 0)$, $(0, j)$, $1 \le i \le n$, $1 \le j \le m$ (the optimal strategies at the boundary points involving essentially single-attribute relations are easily computable [Hevner and Yao 1979]), then the strategies at (y_1, y_2), $y_1 + y_2 = t + 1$ can easily be computed using (6.1)–(6.5). Starting from $t = 2$, we progress to $t = m + n$ when the optimal strategy for the query is obtained. This operation can be shown to take $O(mn)$ time [Yu et al. 1979].

Example 6.4. Given $|A_1| \le |A_2| \le |IA|, |IB| \le |B_1|$, and $m = 2$, $n = 1$; see Figure 14.

(1) For the points on the A axis, the optimal strategies OPTS(k, 0, 0, A_k), OPTS(k, 0, IA, A_k), and OPTS(k, 0, I, I), where $k = 1$ or 2, are calculated as follows:

OPTS(k, 0, 0, A_k):

A_1, $k = 1$,

$A_1 \rightarrow A_2$, $k = 2$

OPTS(k, 0, IA, A_k):

$\min\{A_1 \rightarrow IA \rightarrow A_1, IA \rightarrow A_1\}$, $k = 1$,

$\min\{A_1 \rightarrow A_2 \rightarrow IA \rightarrow A_2,$

$A_1 \rightarrow IA \rightarrow A_2\}$, $k = 2$.

OPTS(k, 0, I, I):

$A_1 \rightarrow I$, $k = 1$,

$A_1 \rightarrow A_2 \rightarrow I$, $k = 2$.

(2) For the points on the B axis, the optimal strategies OPTS(0, 1, 0, B_1), OPTS(0, 1, IB, B_1), and OPTS(0, 1, I, I) are calculated as follows:

OPTS(0, 1, 0, B_1): B_1,

OPTS(0, 1, IB, B_1): $IB \rightarrow B_1$,

OPTS(0, 1, I, I):

$\min\{IB \rightarrow B_1 \rightarrow I, B_1 \rightarrow I\}$.

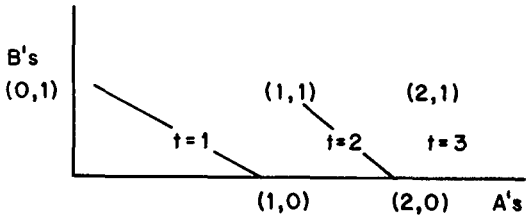


Figure 14. Example 6.4.

Then, all points (x_1, x_2) satisfying $x_i \geq 1$, $x_1 + x_2 = 2$ are located. In our example, (1, 1) is the only point. The three optimal strategies to be considered are OPTS(1, 1, I, A_1), OPTS(1, 1, I, I), and OPTS(1, 1, I, B_1).

By (6.1) and 6.2),

$$\text{OPTS}(1, 1, I, A_1) \quad \text{is} \quad A_1 \leftarrow \text{OPTS}(0, 1, I, I)$$

and

$$\text{OPTS}(1, 1, I, B_1) \quad \text{is} \quad B_1 \leftarrow \text{OPTS}(1, 0, I, I).$$

By (6.3), and (6.4) and (6.5), OPTS(1, 1, I, I) is the minimal cost strategy among the following three strategies:

$$I \leftarrow \text{OPTS}(1, 1, I, A_1),$$

$$I \leftarrow \text{OPTS}(1, 1, I, B_1),$$

and

$$I \leftarrow \begin{cases} \min\{\text{OPTS}(1,0,0,A_1), \\ \quad \text{OPTS}(1,0,I,A_1)\}, \\ \min\{\text{OPTS}(0,1,0,B_1), \\ \quad \text{OPTS}(0,1,I,B_1)\}. \end{cases}$$

There is only one point (2, 1) satisfying $x_i \geq 1$, $x_1 + x_2 = 3$. The three optimal strategies at (2, 1) are calculated by (6.1)–(6.5). They are as follows:

OPTS(2, 1, I, A_2):

$$A_2 \leftarrow \min\{\text{OPTS}(1, 1, I, I), \quad \text{OPTS}(1, 1, I, A_1)\},$$

OPTS(2, 1, I, B_1):

$$B_1 \leftarrow \text{OPTS}(2, 0, I, I),$$

and

OPTS(2, 1, I, I):

$$I \leftarrow \begin{cases} \min\{I \leftarrow \text{OPTS}(2, 1, I, A_2), \\ \quad I \leftarrow \text{OPTS}(2, 1, I, B_1), \\ \quad \begin{cases} \min\{\text{OPTS}(2, 0, 0, A_2), \\ \quad \text{OPTS}(2, 0, IA, A_2)\} \\ \min\{\text{OPTS}(0, 1, 0, B_1), \\ \quad \text{OPTS}(0, 1, IB, B_1)\}. \end{cases} \end{cases}$$

If the minimum cost strategy to fully reduce some relation is sought, the answer is then

$$\min\{\text{OPT}(2,1,I,A_2), \text{OPT}(2,1,I,B_1), \text{OPT}(2,1,I,I)\}. \quad \square$$

The algorithm can be generalized to obtain optimal strategies to fully reduce a relation for tree queries (see Chiu and Ho [1980] and Yu et al. [1979]). However, the algorithm runs in exponential time.

7. HEURISTICS ALGORITHMS BASED ON SEMIJOINS

Two query-processing algorithms using semijoins are discussed in this section. They assume that one copy of each relation referred to by the query has been selected and then the reduction and the assembly phases are carried out. The *cost* of a semijoin $X - A \rightarrow Y$ is defined to be the cost of transferring $X.A$ from the site containing X to the site containing Y (if the two sites are identical, the cost is zero). The *benefit* of the semijoin is the size of Y before the operation minus the size of Y after the operation. A semijoin is *profitable* if its cost is less than its benefit.

7.1 The SDD-1 Query-Processing Algorithm and Its Enhancements [Bernstein et al. 1981; Goodman et al. 1979]

The reduction phase is very simple; it identifies all possible semijoins between any two relations. The cost and the benefit of each semijoin are estimated. A profitable semijoin having the smallest cost is then chosen. (In one of the two papers, the semijoin

having the highest (benefit − cost) is selected.) The costs and the benefits of those semijoins that can be affected by the execution of the chosen semijoin are updated, and another semijoin is considered. The process is repeated until no profitable semijoin can be found. Some details are as follows.

First, all local reductions using selections and projections are performed. Semijoins within the same site can also be executed to reduce the sizes of relations. Then all possible semijoins across sites are identified. As pointed out earlier, after renaming attributes, all semijoins are of the form $R_i \rightarrow A_k \rightarrow R_j$, because semijoins of the form $R_i.A_k \rightarrow R_j.A_l$ are neither stated nor implied by the qualification of the query. For each such semijoin across sites, the cost and the benefit are estimated to be $p_i \mid A_k \mid w$ and $\mid R_j \mid w_j(1 - p_i)$, respectively, where p_i is the selectivity of R_i on attribute A_k, $\mid A_k \mid$ is the cardinality of A_k, w is the average width of a value in A_k, w_j is the average width of a tuple in relation R_j, and $\mid R_j \mid$ is the number of tuples of the relation. The semijoin is profitable if $p_i \mid A_k \mid w < \mid R_j \mid w_j(1 - p_i)$. After identifying all profitable semijoins, the semijoin with least cost is selected to be the first semijoin to be executed, for example, $R_r — A_s \rightarrow R_t$. (The second version of SDD-1 selects the semijoin which maximizes (benefit − cost).) This semijoin is not executed until the entire sequence of semijoins and the assembly site are chosen. In spite of not executing this semijoin immediately, its effect on the relation R_t is estimated. Specifically, the benefits and the costs of semijoins from R_t to other relations have to be updated, due to expected reduction of R_t. After the update is performed, the next semijoin to be executed is chosen with the same criterion: that is, in the first version, the semijoin that has the least cost and is still profitable; and in the second version, the semijoin that has the largest (benefit − cost) and is still profitable. This process is repeated until all possible profitable semijoins have been exhausted.

The assembly phase consists of selecting, among all the sites, the site to which the transmission of all the relations referred to by the query incurs the minimum cost. The site is chosen to be the one containing the largest amount of data after the reduction phase so that the sum of the amount of data transferred from other sites will be minimum. After selecting the assembly site, it may be possible to discard some useless semijoins. If, for example, relation R resides in the assembly site and R is scheduled to be reduced by a semijoin, but is not used to reduce other relations after the scheduled execution of the semijoin, then since R need not be moved to another site during the assembly phase, the semijoin on R is useless and should therefore be discarded.

The operations generated by the SDD-1 algorithm can be improved in the following ways [Yu et al. 1983]. Certain relations that are involved in the execution of semijoins need not be sent to the assembly site for further processing, and therefore both communication cost and local processing cost are saved. Furthermore, semijoins involving these relations can either be eliminated or replaced by other semijoins, yielding a smaller communication cost. Some details are provided below.

When a semijoin, for example, $R_i — X \rightarrow R_j$ is executed, not only is R_j reduced, but in some situations, the contents of R_i are completely incorporated into the resulting R_j so that R_i will not be needed for processing of the query. More precisely, let $J(R_i)$ be the joining attributes of R_i. If (i) $X = J(R_i)$ (which implies $J(R_i) \subseteq J(R_j)$) and (ii) the target of the query either does not contain any attribute of R_i or is equivalent to one without any attribute of R_i, R_i can be eliminated from further consideration after executing the semijoin. In Figure 15a, A_k is the only joining attribute of R_i and the semijoin is $R_i — A_k \rightarrow R_j$. Thus if the semijoin executed is $R_i — A_k \rightarrow R_j$, Condition (i) is satisfied. If the target of the query is that given in Figure 15b, which does not contain R_i, or that given in Figure 15c, which can be transformed to one not containing R_i, then R_i can be eliminated after executing the semijoin. Note that Condition (i) is precisely one of the two key steps in determining whether a given query is a tree query. When it is satisfied, the part of the query containing R_i is a tree (sub)query. In Figure 15a, $R_i — A_k — R_j$ is a tree subquery, which permits R_j to be

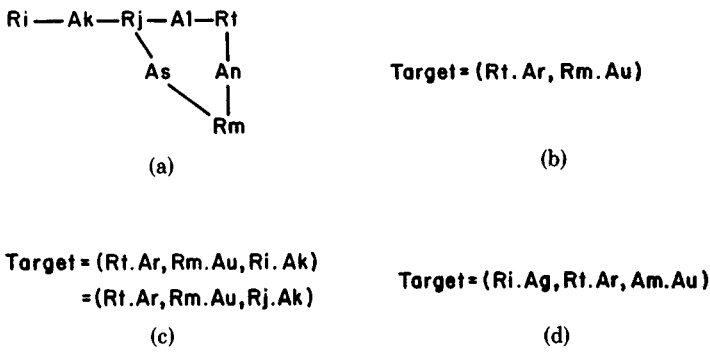


Figure 15. Possibility of eliminating R_i after executing the semijoin $R_i — A_k \rightarrow R_j$: (a) query graph; (b) target; (c) another target; (d) another target.

fully reduced by R_i with respect to it. On the other hand, Condition (i) is not satisfied by semijoins involving any two of the three relations $\{R_j, R_t, R_m\}$ in Figure 15a, for an obvious reason: The subquery involving the three relations is cyclic. Figure 15d shows a target that cannot be transformed into an equivalent target without the relation R_t. Thus considering the target given in Figure 15d, R_t cannot be eliminated even if Condition (i) is satisfied. The satisfaction of both Conditions (i) and (ii) allows R_i to be eliminated after the execution of the semijoin.

As pointed out in the last paragraph, SDD-1 does not recognize that certain relations involved in previously executed semijoins are not needed for further processing and can be eliminated, and that semijoins involving these relations can still be generated subsequently. It turns out [Yu et al. 1983] that any semijoin involving any such disposable relation can always be replaced by another semijoin such that the cost of the new strategy is not higher than that of the original strategy. The replacement procedure begins by letting the semijoin be replaced by $R_i — A \rightarrow R_j$, where R_i or R_j or both relations can be eliminated. R_i will be replaced by R_r, where, if R_i cannot be eliminated, R_r is R_i; otherwise, there exists a sequence of semijoins such that $R_i \rightarrow R_{i1}$ causes R_i to be eliminated, $R_{ij} \rightarrow R_{ij+1}$ causes R_{ij} to be eliminated, $1 \leq j \leq t$, R_{it+1} is not eliminated, and R_r is R_{it+1}. If the relation replacing R_i is denoted by $\text{Repl}(R_i)$, the same replacement procedure applies to R_j. In the example of Figure 16, no relation is eliminated initially; thus $\text{Repl}(R_i) = R_i$, $1 \leq i \leq 4$. After the first semijoin $R_1 — C \rightarrow R_2$, R_1 is eliminated and therefore $\text{Repl}(R_1) = R_2$, as shown in Figure 16b. If the next semijoin is $S_1: R_1 — C \rightarrow R_4$, then the replacement semijoin is $S_2: R_2 — C \rightarrow R_4$. Since R_1 was used to reduce R_2, it is clear that $R_1(C)$ in semijoin S_1 contains $R_2(C)$ in semijoin S_2, and therefore $\text{cost}(S_2) \leq \text{cost}(S_1)$. Furthermore, R_4, which is reduced by semijoin S_1, contains R_4, which is reduced by semijoin S_2. Any semijoin thus originating from the latter R_4 has a smaller cost than the corresponding semijoin originating from the former R_4, and if no further semijoin is executed on R_4, the cost to send the latter relation to the assembly site is smaller than that to send the former relation to the same destination.

Instead of having the next semijoin be S_1, the next semijoin is $S_3: R_4 — C \rightarrow R_1$. It is then replaced by $S_4: R_4 — C \rightarrow R_2$. Since R_1 is not needed for processing the query, the semijoin S_3 is not a useful operation. And since $R_1(C)$ after executing S_3 contains $R_2(C)$ after executing S_4, any semijoin originating from $R_1(C)$ will be more costly than the corresponding one from $R_2(C)$. Thus in both cases, replacing R_1 by R_2 yields a better strategy.

Figure 16c–e shows that after executing some other semijoins other relations are eliminated, and defines the relations that should be replaced by other particular relations at each stage.

$$Q = \{R_2.D \mid (R_1.C = R_2.C) \text{ and } (R_4.C = R_1.C) \text{ and } (R_2.D = R_3.D)\}$$

(a) Initially, no relation is eliminated.

(b) After execution of the semijoin $R_1 - C \rightarrow R_2$,
R_1 is eliminated.
$\text{Repl}(R_1) = R_2$;
$\text{Repl}(R_2) = R_2$;
$\text{Repl}(R_3) = R_3$;
$\text{Repl}(R_4) = R_4$.

(c) After $R_2 - C \rightarrow R_4$,
there is no change, that is,
$\text{Repl}(R_1) = R_2$;
$\text{Repl}(R_2) = R_2$;
$\text{Repl}(R_3) = R_3$;
$\text{Repl}(R_4) = R_4$.

(d) After $R_4 - C \rightarrow R_2$,
R_4 is eliminated.
$\text{Repl}(R_1) = R_2$;
$\text{Repl}(R_2) = R_2$;
$\text{Repl}(R_3) = R_3$;
$\text{Repl}(R_4) = R_2$.

(e) After $R_2 - D \rightarrow R_3$,
R_2 is eliminated.
$\text{Repl}(R_1) = R_3$;
$\text{Repl}(R_2) = R_3$;
$\text{Repl}(R_3) = R_3$;
$\text{Repl}(R_4) = R_3$.

Figure 16. Replacing an eliminated relation by another relation.

7.2 The General Algorithm in Apers et al. [1983]

Apers et al. [1983] present an algorithm that is a generalization of the optimal algorithm given in Section 5 to process simple queries. The strategy constructed by the algorithm is a union of n substrategies, one for each relation, where n is the number of relations referenced by the query. Consider a relation R of the query. Let A be a joining attribute of R. An optimal method is sought to reduce and send R to the result site using semijoins on attribute A only. Given that $R_{i1}, R_{i2}, \ldots, R_{it}$, $t \leq n$, be the relations of the query having joining attribute A such that their projections on A are arranged in ascending order of size, that is, $|R_{i1}(A)| \leq |R_{i2}(A)| \leq \cdots \leq |R_{it}(A)|$, and supposing that R is R_{ik} for some $1 \leq k \leq t$, then, by the result given in Section 5, single-attribute relations in optimal strategies to send R_{ik} to the result site should be in ascending order of size. Thus the *candidate schedules on attribute A* to send the entire relation R_{ik} to the result site are

(A) $R_{i1}(A) - A \rightarrow R_{ik} \rightarrow$
$R_{i1}(A) - A \rightarrow R_{i2}(A) - A$
$\rightarrow R_{ik} \rightarrow$
$\vdots$
$R_{i1}(A) - A \rightarrow R_{i2}(A) - A \rightarrow \cdots$
$\rightarrow R_{ik-1}(A) - A \rightarrow R_{ik} \rightarrow$

and

(B) $R_{i1}(A) - A \rightarrow R_{i2}(A) - A \rightarrow \cdots$
$\rightarrow R_{ik-1}(A) - A \rightarrow R_{ik}(A) -$
$A \rightarrow R_{ik+1}(A) - A \rightarrow R_{ik} \rightarrow$
$\vdots$
$R_{i1}(A) - A \rightarrow R_{i2}(A) - A \rightarrow \cdots$
$\rightarrow R_{ik-1}(A) - A \rightarrow R_{ik}(A) - A \rightarrow$
$\cdots \rightarrow R_{it}(A) - A \rightarrow R_{ik} \rightarrow$

where the last data transfer in each candidate schedule "$R_{ik} \rightarrow$" sends the entire relation R_{ik} to the result site. In each strategy in (A), R_{ik} occurs once, while in each strategy in (B), both $R_{ik}(A)$ and R_{ik} occur once. In the latter situation, there are two possible cases for each schedule, one having $R_{ik}(A)$ as given in the figure and the other leaving out $R_{ik}(A)$. The minimum cost schedule among all these schedules is chosen and is denoted as the *best strategy to reduce R_{ik} on attribute A*. This procedure is repeated for each joining attribute of each relation R. Let $\text{BST}_1, \text{BST}_2, \ldots, \text{BST}_p$ be the set of all the best strategies to reduce R on the joining attributes of relation R, where p is the number of attributes of relation R appearing in the qualification of the query. Assume $\text{cost}(\text{BST}_1) \leq \text{cost}(\text{BST}_2) \leq \cdots \leq \text{cost}(\text{BST}_p)$, where $\text{cost}(\text{BST}_i)$ is the cost of the best strategy BST_i to reduce R on a certain attribute of

R and send the resulting R to some other site. $BST_1, BST_2, \ldots, BST_q$ are combined to form a strategy to reduce R, $1 \leq q \leq p$. As q varies from 1 to p, p combined strategies are formed. The combined strategy having the smallest cost is the *least cost strategy* to reduce R. A similar strategy for each relation is produced at the reduction phase.

In the assembly phase, the reduced relations are then sent to the result site to produce the answer as in the following example.

Example 7.1. Suppose that R_1, R_2, R_3, and R_4 are four relations, each residing in a different site. Let $Q = \{(R_3.X, R_4.Y) \mid (R_1.A = R_2.A) \text{ AND } (R_2.A = R_4.A) \text{ AND } (R_3.B = R_1.B)\}$.

Figure 17a describes the size of the domain of each joining attribute, the size of each relation, and the selectivity of each relation on each joining attribute.

Figure 17b presents the candidate schedules for attributes A and B. Assume that $c_0 = 0$ and $c_1 = 1$. The best strategies for relation R_1 on attributes A and B are then obtained. They are $BST_1: R_3 — B \rightarrow R_1 \rightarrow$ and $BST_2: R_4 — A \rightarrow R_1 \rightarrow$, where $\text{cost}(BST_1) = 400$ and $\text{cost}(BST_2) = 440$. The combined strategies to reduce R_1 are

$$R_3 — B \rightarrow R_1 \rightarrow$$

and

$$\begin{array}{l} R_3 — B \rightarrow R_1 \rightarrow \\ \qquad\qquad\quad | \\ \qquad\qquad\quad A \\ \qquad\qquad\quad | \\ \qquad\qquad\quad R_4 \end{array}$$

The least cost strategy to reduce R_1 is then selected from the above two combined strategies, and the least cost strategies of R_2, R_3, and R_4 are selected by a similar process. Figure 17d shows the least cost strategies for all four different relations.

Figure 17e gives the final strategy to answer the query. □

7.3 Better Semijoin Sequence

Each of the two algorithms above constructs a semijoin strategy to answer a given query. However, each of the constructed semijoin strategies can sometimes be improved. In Luk and Luk [1980], a polynomial time algorithm is presented to transform a given semijoin strategy (produced by some heuristic) into an equivalent strategy such that each semijoin in the former strategy corresponds to a semijoin in the latter strategy and incurs neither higher cost nor lower benefit. (The enhancements in SDD-1 given in Section 7.1 are applicable when one or more relations are eliminated. Here, the procedure is applicable even if no relation is eliminated.)

For example, the node R_1 in substrategy $R_2 — AB \rightarrow R_1 \leftarrow B — R_3$ has in-degree >1, and the label in one semijoin is a subset of the label in the other. Satisfying the conditions above guarantees that a better strategy can be obtained. $R_3 — B \rightarrow R_2 — AB \rightarrow R_1$ is an example of such a strategy, because although the semijoin $R_3 — B \rightarrow X$ (X is R_1 in the former strategy and is R_2 in the latter strategy) is executed with the same cost in both strategies, the semijoin $R_2 — AB \rightarrow R_1$ is executed with a smaller cost in the latter strategy. This process can be applied to R_3, if it should satisfy the above conditions. The algorithm by Luk and Luk [1980] scans a given strategy and identifies the situations in which a node has either in-degree >1 or out-degree >1, and checks whether the label in one semijoin involving the node is a subset of that in another semijoin involving the same node. When such a situation is detected, the algorithm replaces the substrategy by a better one. This process is applied to the preceding nodes recursively until no such situation exists.

8. ALGORITHMS BASED ON JOINS

Although the use of semijoins reduces the amount of data transfer and is a valuable tool, it is not always superior to the use of joins only. One reason is that for certain networks, the number of messages exchanged rather than the amount of data transferred may be the dominating factor. Additional messages may be generated when semijoins are employed. Another reason is that local processing costs can be significant, and since SDD-1 and related

	Size	Selectivity A	Selectivity B
R_1	1200	0.2	0.5
R_2	600	0.6	
R_3	1200		0.25
R_4	2000	0.2	

Cardinality of $A = 1000$.
Cardinality of $B = 400$.
$c_0 = 0, c_1 = 1$.

Each distinct value in A and each distinct value in B have unit length.

$$Q = \{(R_3.X, R_4.Y) \mid (R_1.A = R_2.A) \text{ AND } (R_2.A = R_4.A) \text{ AND } (R_1.B = R_3.B)\}$$

(a)

A: cost		B: cost	
200	$R_1 - A \rightarrow$	100	$R_3 - B \rightarrow$
200 + 40	$R_1 - A \rightarrow R_4 - A \rightarrow$	100 + 50	$R_3 - B \rightarrow R_1 - B \rightarrow$
200 + 40 + 24	$R_1 - A \rightarrow R_4 - A \rightarrow R_2 - A \rightarrow$		

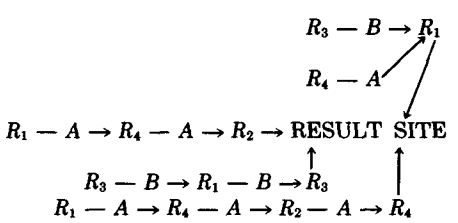
(b)

	A	B
R_1: cost		
200 + 240	$R_4 - A \rightarrow R_1$	
cost		
100 + 300		$R_3 - B \rightarrow R_1 \rightarrow$
R_2:		
200 + 40 + 24	$R_1 - A \rightarrow R_4 - A \rightarrow R_2 \rightarrow$	
R_3:		
100 + 50 + 600		$R_3 - B \rightarrow R_1 - B \rightarrow R_3 \rightarrow$
R_4:		
200 + 40 + 24 + 240	$R_1 - A \rightarrow R_4 - A \rightarrow R_2 - A \rightarrow R_4 \rightarrow$	

(c)

R_1: $R_3 - B \rightarrow R_1 \rightarrow$
$R_4 - A -$
R_2: $R_1 - A \rightarrow R_4 - A \rightarrow R_2 \rightarrow$
R_3: $R_3 - B \rightarrow R_1 - B \rightarrow R_3 \rightarrow$
R_4: $R_1 - A \rightarrow R_4 - A \rightarrow R_2 - A \rightarrow R_4 \rightarrow$

(d)



(e)

Figure 17. An example illustrating the general algorithm in Apers et al. [1983]. (a) Size of domain of each joining attribute, size of each relation, and selectivity of each relation on each joining attribute; (b) candidate schedules for each joining attribute; (c) best strategies for reducing each relation on each of its attributes; (d) least cost strategies of different relations; (e) strategy for answering the query.

algorithms ignore these costs, the actual processing cost of strategies based on these algorithms can be high. Last, although semijoins can be executed in parallel, the minimization of response time using semijoins is complicated [Apers et al. 1983]. Several algorithms using joins are studied below.

8.1 Enumerative Algorithms

8.1.1 *Algorithm in Epstein and Stonebraker* [1980]

The algorithm first partitions the set of relations in the query into two complementary groups, G_1 and G_2, where G_1 has at least two relations and G_2 has zero or more relations. Substrategy for the relations in G_1 is next obtained by designating the site containing the largest relation as the result site and sending all other relations in G_1 to it. It seeks the minimal cost substrategy by a recursive call for the relations in $G_2 \cup \{R\}$, where R is the resulting relation obtained from the those relations in G_1. All possible combinations of G_1 and G_2 are considered to obtain the minimal strategy.

Should relations R_1, R_2, and R_3 reside in different sites and a query asks for the join of these three relations, the algorithm will first partitions R_1, R_2, and R_3 into $\{\{R_1, R_2\}, \{R_3\}\}$. Then the minimal cost substrategy for $\{R_1, R_2\}$ is constructed by sending the smaller of the two relations R_1 and R_2 to the other. The relation obtained by joining R_1 with R_2, for example, T_1, is added to the second group and the minimal cost substrategy for $\{T_1, R_3\}$ is sought. A strategy for the joins of R_1, R_2, and R_3 is then obtained. The same process is repeated for $\{\{R_1, R_3\}, \{R_2\}\}$, $\{\{R_2, R_3\}, \{R_1\}\}$, and $\{\{R_1, R_2, R_3\}, \{\ \}\}$. At the end, the optimal strategy for the query is obtained.

In general, when the natural join of n relations is sought, the exhaustive enumerative search algorithm will scan through $e(n)$ strategies, where

$$e(1) = 1,$$

$$e(2) = 1,$$

$$e(n) = \sum_{i=2}^{n} \binom{n}{i} * e(n - i + 1),$$

where $\binom{n}{i}$ stands for the number of different combinations for the first group having i relations, and $e(n - i + 1)$ stands for the number of substrategies that the recursive call will scan through if the first group has i relations.

By leaving out the lower order terms (i.e., $i \geq 3$) in the above expression, $e(n) \geq \binom{n}{2} e(n-1).\ e(4) = 29;\ e(5) \geq 10 * 29 = 290;$ $e(6) \geq 15 \times 290 = 435$. Thus $e(n)$ grows very rapidly, although some of the strategies are degenerate (i.e., certain subsets of relations may not be joined, but strategies involving these subsets of relations are enumerated).

8.1.2 *R** [Williams et al. 1981]

As in Epstein and Stonebraker [1980], R* enumerates many strategies and chooses the one with the least cost. However, many more alternatives are considered in R*. If a relation is replicated, the choice of the appropriate copies of the relation to be used for processing the query has a significant effect on the cost; the sequence in which the operations are performed is also important. For example, the cost of the strategy ((R_1 joined with R_2) joined with R_3) differs from that of the strategy (R_1 joined with (R_2 joined with R_3)). Even the join between two different relations R_1 and R_2, situated at distinct sites S_1 and S_2, respectively, can be performed in several ways, resulting in different costs, for example:

(i) Send R_1 to site S_2 and join with R_2 there.
(ii) Send R_2 to site S_1 and join with R_1 there.
(iii) Send both relations R_1 and R_2 to a different site S_3 and join them there.
(iv) For each tuple of R_1 transmitted to S_2, send the matching tuples of R_2 to S_1.
(v) The same as (iv), with the roles of R_1 and R_2 reversed.

Many strategies are thus evaluated by R*, which takes into consideration both local processing cost and data communication cost. Although enumerating all these strategies for a query can be costly, this approach can be worthwhile if the query is frequently executed. Such an approach is

also taken in centralized databases [Griffiths Selinger et al. 1979].

8.2 Nonenumerative Algorithms

8.2.1 *Algorithm in Baldissera et al.* [1979]

The algorithm in Baldissera et al. [1979] accepts only tree queries. It decomposes a query into chain queries and solves them to obtain the answer. A *chain query* is a query whose query-graph or equivalent query-graph is a chain. A *nonchain query* is a query for which none of its equivalent query graphs is a chain.

Suppose that a chain query with a node designated as root is given. The algorithm finds the assembly site, which is the site with the maximum number of data referenced by the query. Then the algorithm repeats the following process until the answer to the chain query is obtained. Starting from a leaf, the algorithm checks whether joining the leaf with its parent first and then sending the result to the assembly site incurs less cost than sending the two relations directly to the assembly site and performing the join there. If the former strategy is less costly, then the leaf node and its parent are merged to form a temporary relation, and the query graph is modified by replacing the part of the graph connecting the two relations by the newly created relation. Otherwise, the leaf node is sent to the assembly site and is eliminated from the query graph. This process is repeated over the modified query graph. When two relations are joined, the algorithm sends the smaller relation to the site containing the larger relation and merges them.

Figure 18c gives an example of a chain query. A decision has to be made whether to merge the leaf R_5 and its parent R_{34} to form a new relation R_{345} or to send R_5 and R_{34} directly to the root. The choice with the lower transmission cost is selected.

If the query graph is a tree but not a chain, and R is the root, adjacent to k nodes, the following two cases arise:

Case 1. $k > 1$. The tree is decomposed into k subtrees, with each subtree containing R as the root. R is the only node in common between the subtrees. For example, the subtrees of Figure 18a are given in Figure 18d. The subtree with the smallest number of nodes is first selected for processing. If the subtree is a chain, the previous procedure is applied to the subquery corresponding to the subtree, and the root is modified and incorporated into the original tree. For example, processing the chain in Figure 18d and incorporating the modified root into the original tree yields the modified query-graph in Figure 18b. If the selected subtree is not a chain, then the present procedure is applied recursively to the subtree.

Case 2. $k = 1$. Let the direct descendant of R be r. If r has two or more direct descendants, then the subtree with root r is identical to Case 1 and the same procedure is applied to it; otherwise, process in Case 2 is applied.

Example 8.1. Given a tree query with root R_1 as shown in Figure 18a, since R_1 is adjacent to two nodes, the tree is decomposed into two subtrees, as shown in Figure 18d. Let the corresponding subqueries be Q_1 and Q_2. Since Q_1 is a chain query, it is processed as described above. Relations R_1 and R_2 are merged to form the modified root R_{12}, which is incorporated into the original tree to form the modified query graph as shown in Figure 18b. This modified query graph has one subtree only, and thus it belongs to Case 2. The direct descendant of R_{12}, R_4, has two direct descendants. Thus the subtree with root R_4 is decomposed into two subtrees, namely $R_4 - C - R_3$ and $R_4 - D - R_5$. Suppose that the former subtree is selected for processing. R_3 and R_4 are merged to form R_{34}, and the modified query-graph is that shown in Figure 18c. Since this is a chain query, the procedure for processing chain queries is invoked. □

8.2.2 *The INGRES Algorithm* [Epstein et al. 1978]

A given query is decomposed into a sequence of subqueries $Q_1, Q_2, \ldots, Q_p$ with at most one variable in common between two consecutive subqueries, as in Wong and

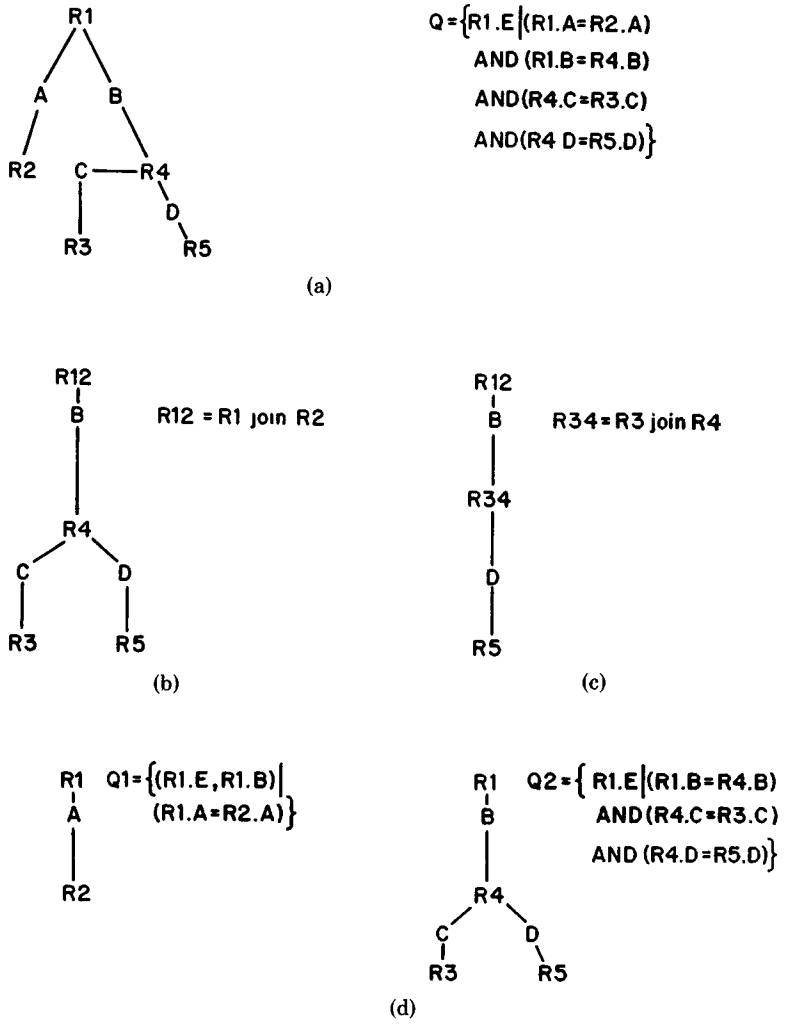
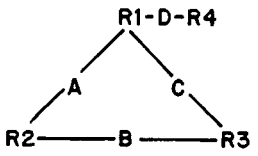


Figure 18. Query processing in Baldissera et al. [1979].

Youssefi [1976]. Each subquery is irreducible. A query is *irreducible* if and only if its query-graph is a chain with two nodes or a cycle with k nodes and all its equivalent query-graphs have a cycle with the same k nodes, where $k \geq 3$. For example, given a query as shown in Figure 19a, the algorithm decomposes the query into two irreducible subqueries Q_1 followed by Q_2, as shown in Figure 19b. Q_1 is processed and the result is incorporated into the query graph of Q_2, which is then processed.

Distributed INGRES [Epstein et al. 1978] considers both data communication and local processing costs and allows relations to be fragmented in various sites (see Section 9). For ease of presentation, it is assumed that relations are not fragmented, and only the data communication cost is considered. If a subquery is a chain with two nodes, say R_x and R_y, then either R_x is sent to the site containing R_y, or R_y is sent to the site containing R_x, depending on which strategy incurs less cost. If the subquery is a cyclic query, then a decision has to be made whether to process the entire subquery at once or subdivide it into pieces. The subquery is subdivided if it

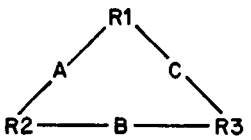


$$Q = \{R4.E \mid (R1.A = R2.A) \text{ AND } (R1.D = R4.D) \text{ AND } (R2.B = R3.B) \text{ AND } (R3.C = R1.C)\}$$

(a)

$$Q1 = \{R1.D \mid (R1.A = R2.A) \text{ AND } (R2.B = R3.B) \text{ AND } (R1.C = R3.C)\}$$

$$Q2 = \{R4.E \mid (R123.D = R4.D)\}$$



where R123 is the result of processing Q1

(b)

$$|R1| \leq |R2| \leq |R3|$$

process at once

(c)

R1 → R2 → R3
R1 → R3 → R2
R2 → R3 → R1

subdivide to a sequence of operations

(d)

Figure 19. Query processing in Epstein et al. [1978].

results in a lower cost. As an example, given three relations R_1, R_2, and R_3 with $|R_1| \leq |R_2| \leq |R_3|$, $w_1 = w_2 = w_3$, and each relation residing in a different site, $|R_i|$ is the number of tuples of R_i and w_i is the average tuple width of R_i. Figure 19c illustrates the strategy for processing the subquery Q_1 at once with minimum cost, while Figure 19d shows all possible strategies for subdividing Q_1 into pieces. The strategy in Figure 19c means that R_1 and R_2 are sent to the site containing R_3, and the answer to Q_1 is produced there. The first strategy, in Figure 19d, for instance, is interpreted as follows: (i) R_1 is sent to the site containing R_2 to perform the join of R_1 with R_2; (ii) the resulting relation, let us say R_{12}, is sent to the site containing R_3 to perform the join with R_3. The cost of the minimal strategy in Figure 19d is compared with that of the strategy in Figure 19c. If the former strategy has less cost, Q_1 is then subdivided into the pieces as given by the minimal strategy; otherwise, Q_1 is processed at once.

It is important to point out that both the query processing algorithms in distributed INGRES and R* take into consideration the local processing cost as well as the data transmission cost. Distributed INGRES also provides a different algorithm to optimize the response time of a query in a broadcasting system.

Wong [1981] suggested decomposing a given query into a sequence of subqueries that contain at most one join and possibly

some projections and selections. His method emphasizes maximizing parallelism by making use of redundancy of data.

9. FRAGMENT PROCESSING

A relation can be viewed as a matrix where the rows stand for tuples and the columns stand for attributes. A horizontal fragment of a relation is a subset of the rows of the matrix. It is obtained by applying a select operation on the relation. Sometimes a horizontal fragment is accessed frequently in one site, while another horizontal fragment is referenced frequently in another site. Thus it may be beneficial to assign fragments to sites according to their reference locality. A vertical fragment of a relation is a subset of the columns of the relation and is constructed by using the projection operation on the relation. In this section, we restrict our discussion to horizontal fragments.

In Goodman et al. [1979], a query that refers to fragmented relations is first decomposed into subqueries. The SDD-1 algorithm described in Section 7 is then used to obtain an answer for each subquery. The union of the answers of all the subqueries is the answer to the query.

The following procedure is used to decompose the query into subqueries: For a given query, (1) find all the fragmented relations referenced by the query, say F, G, ..., and H; (2) for each combination of fragments F_i, G_j, ..., and H_k, where F_i, G_j, ..., and H_k are the fragments of F, G, ..., and H, respectively, construct a subquery by replacing F, G, ..., and H in the query by F_i, G_j, ..., and H_k, respectively. Thus the number of subqueries is equal to the product of the numbers of fragments of the referenced relations.

Example 9.1. Let $F = \{F_1, F_2\}$ and $G = \{G_1, G_2\}$ be two fragmented relations.

Let $Q = \{F.B \mid F.A = G.A\}$ be a query.

Query Q is decomposed by the above procedure into the subqueries:

$$Q_1 = \{F_1.B \mid F_1.A = G_1.A\},$$

$$Q_2 = \{F_2.B \mid F_2.A = G_1.A\},$$

$$Q_3 = \{F_1.B \mid F_1.A = G_2.A\},$$

$$Q_4 = \{F_2.B \mid F_2.A = G_2.A\}.$$

The four subqueries are then individually evaluated by SDD-1's query processing algorithm, and the union of the answers is the final answer.

A semijoin in a fragmented database environment will fall into one of the following three types: F–F, R–F, or R–R [Chang 1982b]. An F–F semijoin is one in which both the sending and the reduced relations are fragments; An R–F's sending relation is a whole relation, but the reduced relation is a fragment; in an R–R semijoin, both relations are whole relations. Thus a query-processing algorithm in a fragmented database environment can be classified into three categories: F–F semijoin-based algorithm, R–F semijoin-based algorithm, and R–R semijoin-based algorithm.

Version 1 of SDD-1's query-processing algorithm [Goodman et al. 1979] as described in Example 9.1 is an F–F semijoin-based algorithm. An R–F semijoin-based algorithm is introduced in Chang [1982b]. It repeatedly chooses a beneficial R–F semijoin until no beneficial R–F semijoin exists. Then the reduced fragments/relations are sent to the assembly site to produce the answer. The query-processing algorithm in Yu et al. [1983] is an R–R semijoin-based algorithm. For each given semijoin $R_i - A \rightarrow R_j$, where R_i and R_j may or may not be fragmented, it selects a set of sites where the semijoin can be performed with minimum cost.

Another way to process a query referencing fragmented relations follows [Epstein et al. 1978; Stonebraker et al. 1982]: One fragmented relation is chosen, and other fragmented relations referenced by the query are replicated at the sites of the chosen fragmented relation. As an example, let a query reference R_1 and R_2. Suppose that R_1 contains fragments F_{11} at site 1 and F_{12} at site 2, and R_2 contains fragments F_{21} at site 1, F_{22} at site 3, and F_{23} at site 4. The algorithm may then choose R_1 to remain fragmented and replicate R_2 at sites 1 and 2. The latter operation is performed by sending F_{21} to site 2, and sending both F_{22} and F_{23} to sites 1 and 2. After R_2 arrives at the sites, R_2 is joined with F_{11} at site 1 and with F_{12} at site 2. The union of the tuples at the two sites is the final answer. In

Epstein et al. [1978], the relation to remain fragmented is chosen such that the amount of data processed is minimized. In reality, the cost of accessing data depends on the supporting access path. For example, with the use of an index, access could be speeded up significantly. In Yu et al. [1984b], such consideration is given to minimize the cost.

It is clear that this method of processing fragments may require substantial data transfer. We believe that in a realistic environment fragments are not placed arbitrarily, and there are *placement dependencies* between the locations of certain sets of fragmented relations on certain attributes. For example, Students (student-id, course-id, ...) and Courses (course-id, instructor, ...) are two fragmented relations with a fragment of each relation situated at each campus of a university. Since a student usually takes courses only from his or her own campus, the join of a Student fragment in a campus and a Course fragment in a different campus on the attribute course-id is null. In other words, the join of the relations Course and Student can be performed at local sites without data transfer. Formally, if F_{ij} should be a fragment of relation R_i at site j, a *placement dependency between R_1 and R_2 on attribute A* will exist if the join of F_{1k} and F_{2t} on attribute A is null for $k \neq t$.

A query may reference a number of fragmented relations that share placement dependencies among them on a certain set of attributes. It is desirable to determine whether the query can be processed without data transfer [Yu et al. 1984a]. (A dual problem is to determine the placement of fragments such that queries can be processed without data transfer. A solution of the dual problem is given in Wong and Katz [1983].)

First, one seeks two relations (among the referenced relations of the query) that have a placement dependency between them on a certain attribute, that attribute being one of the joining attributes of the two relations. If two such relations cannot be found, the query cannot be processed without data transfer. Otherwise, LP_1 becomes the set containing these two relations, which can be joined together without data transfer, although in practice they need not be the first pair of relations to be operated on. If another relation referenced by the query has a placement dependency with a relation in LP_1 on a set of attributes that is a subset of the set of joining attributes of the query, then it is added to LP_1. This process is repeated until either all relations are added to LP_1, in which case the query can be processed without data transfer or some relations remain and the query cannot be so processed.

If a query references both fragmented and unfragmented relations, then the following condition is sufficient for the query to be processed without data transfer. The fragmented relations (if there are two or more such relations referenced by the query) should satisfy the condition of the algorithm given in the last paragraph, while a copy of the unfragmented relations remains at each of the sites containing the fragmented relations.

10. THE TRANSFORMATION APPROACH

Perhaps a more systematic way to process queries is the transformation approach [Ceri and Pelagatti 1984; Ullman 1980] given as follows. In this approach, there exists a set of rules, where each rule transforms a query expression into an equivalent expression. The idea is to apply these rules repeatedly to obtain an expression that can be evaluated with a small cost.

Typically, the resulting relation after applying a unary operator, like project or select, tends to be smaller than the original relation, while the resulting relation after applying a binary operator, like join or union, can be significantly larger than the original operands. If the operands are in different sites, it will be profitable to reduce their sizes by applying the unary operators while preserving the equivalence of the expressions. For example, joining the relations $R_1(A, B, C)$ and $R_2(B, E, F)$ on attribute B and then projecting the result on the attributes (A, B, E) is equivalent to projecting R_1 on attribute A and B to eliminate C, projecting R_2 on the attributes B and E to eliminate F, and then taking the join of the two reduced relations. If R_1 and

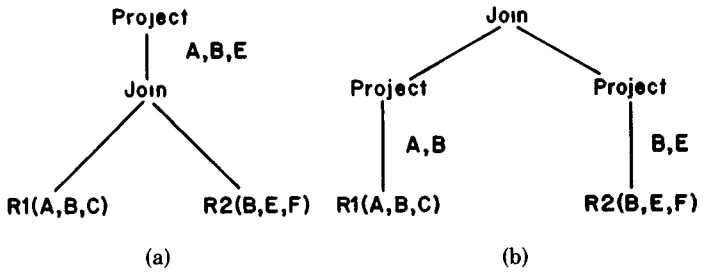


Figure 20. Two equivalent expression trees.

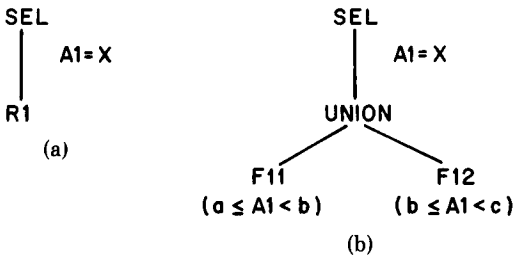


Figure 21. Fragment processing by eliminating unnecessary fragments.

R_2 are in different sites, the latter expression can be evaluated with less data transfer and is therefore preferable. The rule that is applicable in this case is

$$U(R_1 \text{ B } R_2) = U(R_1) \text{ B } U(R_2),$$

where U is a unary operator and B is a binary operator. A complete set of rules and the conditions under which the rules are applicable can be found in Ceri and Pelagatti [1984] and Ullman [1980].

In general, an expression can be represented by an expression tree in which each binary operation between two operands is represented by a subtree where the operands are two nodes whose parent is the operation, and each unary operation on an operand is represented by a subtree with the operation as the parent of the operand (illustrated in Fig. 20a). The application of the rule transforms the expression tree into an equivalent expression tree given in Figure 20b. The strategy is to move the unary operators toward the leaves of the tree as much as possible. The evaluation of the expression tree starts from the leaves toward the root, so that the unary operators can be evaluated as quickly as possible, reducing the original relations to smaller ones. This strategy of reducing the sizes of intermediate relations applies to both centralized and distributed databases. In centralized databases, the intermediate relations move between the secondary and the main memories, while the movement in distributed databases is between the sites. In each case, reducing the sizes of the intermediate results seem logical.

When a relation is fragmented and placed into two or more sites, an expression involving the relation can be rewritten as an expression involving the fragments of the relation. Figure 21 illustrates how relation R_1 is replaced by its fragments F_{11} and F_{12}, where each fragment is defined by a condition on the attribute A_1. In Figure 21, a selection is applied to R_1 on the attribute A_1. Since only one of the fragments F_{11} and F_{12} can satisfy the selection condition $A_1 =$ "X," the expression reduces to a selection of a single fragment. Thus, although the fragments are located at different sites, it is sufficient to perform a selection at the site containing the appropriate fragment; the transfer of the other fragment is not required.

It is easy to see that the earlier approaches are special cases of the transformation approach. For example, a semijoin is used to transform a given expression

involving a join into an equivalent expression including the semijoin operation. A rule in support of this process is

$$R_1 \text{ join } R_2 = R_1 \text{ join } (R_1 \rightarrow R_2).$$

Thus R_2 can be reduced by the semijoin operation before the join with R_1 takes place.

Similarly, the fragment and replicate approach [Epstein et al. 1978] is supported by the rule

$$R_1 \text{ join } \bigcup_i F_{2i} = \bigcup_i (R_1 \text{ join } F_{2i})$$

in which R_1 is joined with each individual fragment of R_2 and then the union is taken rather than assembling all fragments of R_2 into a site and then joining with R_1.

Not only can equivalence of expressions be captured by the rules, but semantic information can also be represented. For example, if a relation should give the facilities of ports and another relation the properties of ships, then the type of ships that can go to their respective type of ports can be expressed as a rule. Thus artificial intelligence techniques may also be applicable in the processing of queries (see, e.g., King [1982]). When the number of rules is large, it is difficult to choose the appropriate sequence of rules to be applied, and it can be a time-consuming process.

11. CONCLUSION

We assume in this paper that a relational database is used, and that queries are expressed in a QUEL-like tuple relational calculus. We did not cover those query-processing algorithms for aggregate queries [Kim 1982; Yu et al. 1984a] and quantified queries [Jarke and Koch 1983; Jarke and Schmit 1982]. We have sketched some of the ideas used in the existing distributed query-processing algorithms: the estimation of the size of intermediate relations, the use of semijoins, the separation of an algorithm based on semijoins into three phases, the properties of tree queries that allow them to be processed rather efficiently, the transformation of cyclic queries into tree queries, the enhancement of semijoin strategies, the enumeration of strategies, and the different ways of handling fragments. The transformation approach can be viewed as a generalization of some of the ideas presented here. We hope that large-scale experiments will be conducted to verify the usefulness of these ideas.

ACKNOWLEDGMENTS

We are grateful to the referees for their suggestions and clarification. One referee in particular has contributed much toward reorganizing the paper to improve readability.

A much abbreviated version of this paper appears in *Topics in Information Science*, Kim, Reiner, and Batory, Eds. Springer Verlag, Berlin and New York, 1985.

REFERENCES

ADIBA, M., CHUPIN, J. C., DEMOLOMBE, R., BIHAN, J. L., AND GARDARIN, G. 1978. Issues in distributed database management systems: A technical overview. In *Proceedings of the 4th International Conference on Very Large Data Bases* (West Berlin, Sept. 13–15). IEEE, New York, pp. 89–110.

APERS, P., HEVNER, A., AND YAO, S. B. 1983. Optimization algorithm for distributed queries. *IEEE Trans Softw Eng. SE-9,* 1 (Jan.), 57–68.

BABB, E. 1979. Implementing a relational database by means of specialized hardware. *ACM Trans. Database Syst. 4,* 1 (Mar.), 1–29.

BALDISSERA, C., BRACEHI, G., AND CERI, S. 1979. A query processing strategy for distributed databases. *EURO IFIP 79,* P. A. Samet, Ed. Elsevier, New York, pp. 667–677.

BEERI, C., FAGIN, R., MAIER, D., MENDELZON, A., AND YANNAKAKIS, M. 1981. Properties of acyclic database schemes. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing* (Milwaukee, Wis., May 11–13). ACM, New York, pp. 355–362.

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

BERNSTEIN, P., AND CHIU, D. 1981. Using semijoins to solve relational queries. *J. ACM 28,* 1 (Jan.), 25–40

BERNSTEIN, P., AND GOODMAN, N. 1979. Full reducers for relational queries using multiattribute semijoins. Tech. Rep., Center for Research in Computing Technology, Harvard Univ., Cambridge, Mass., July.

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

BERNSTEIN, P., GOODMAN, N., WONG, E., REEVE, C., AND ROTHNIE, J. 1981. Query processing in a system for distributed databases (SDD-1). *ACM Trans. Database Syst 6,* 4 (Dec.), 602–625.

BLACK, P., AND LUK, W. 1982. A new heuristic for generating semijoin programs for distributed query processing. In *Proceedings of the IEEE 6th International Computer Software and Application Conference* (Chicago, Ill., Nov. 8–12). IEEE, New York, pp. 581–588.

CERI, S., AND PELAGATTI, G. 1984. *Distributed Databases, Principles and Systems.* McGraw-Hill, New York.

CHANDY, K. 1977. Models of distributed systems. In *Proceedings of the 3rd International Conference on Very Large Data Bases* (Tokyo, Oct. 6–8). IEEE, New York, pp. 105–120.

CHANG, J. 1982a. A heuristic approach to distributed query processing. In *Proceedings of the 8th International Conference on Very Large Data Bases* (Mexico City). VLDB Endowment, Saratoga, Calif., pp. 54–61.

CHANG, J. 1982b. Query processing in a fragmented database environment. Tech. Rep., Bell Laboratories.

CHEN, A. L. P., AND LI, V. O. K. 1983. Properties of optimal semi-join programs for distributed query processing. In *Proceedings of the IEEE 7th International Computer Software and Application Conference* (Chicago, Ill., Nov. 7–11). IEEE, New York, pp. 476–483.

CHEUNG, T. 1981. Two methods of resolution for general equi-join queries in distributed relational database. Tech. Rep., Dept. of Computer Science, Univ. of Ottawa, Ottawa, Ont., Canada.

CHIU, D. 1980. Optimal query interpretation for distributed databases. Ph.D. dissertation, Division of Applied Sciences, Harvard Univ., Cambridge, Mass.

CHIU, D., AND HO, Y. 1980 A methodology for interpreting tree queries into optimal semi-join expressions. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (Santa Monica, Calif., May 14–16). ACM, New York, pp. 169–178.

CODD, E. F. 1970. A relational model for large shared databases. *Commun. ACM 13,* 6 (June), 377–389.

CODD, E. F. 1972. Further normalization of the database relational model. In *Database Systems* Prentice-Hall, Englewood Cliffs, N.J., pp. 33–64.

DATE, C. J. 1977. *An Introduction to Database Systems.* Addison-Wesley, Reading, Mass.

EPSTEIN, R., AND STONEBRAKER, M. 1980. Analysis of distributed database processing strategies. In *Proceedings of the 6th International Conference on Very Large Data Bases* (Montreal, Oct. 1–3). IEEE, New York, pp. 92–101.

EPSTEIN, R., STONEBRAKER, M., AND WONG, E. 1978. Distributed query processing in a relational database system. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (Austin, Tex., May 31–June 2). ACM, New York, pp. 169–180.

FAGIN, R., MENDELZON, A., AND ULLMAN, J. 1980. A simplified universal relation assumption and its properties. Tech. Rep., IBM.

GOODMAN, N., AND SHMUELI, O. 1982a. The tree property is fundamental for query processing. In *proceedings of the ACM Symposium on Principles of Data Systems* (Los Angeles, Calif., Mar. 29–31). ACM, New York.

GOODMAN, N., AND SHMUELI, O. 1982b. Transforming cyclic schemas into trees. In *Proceedings of the ACM Symposium on Principles of Data Systems* (Los Angeles, Calif., Mar. 29–31). ACM, New York.

GOODMAN, N., AND SHMUELI, O. 1983. Syntactic characterization of tree database schema. *J ACM 30,* 4 (Oct.), 767–786.

GOODMAN, N., et al 1979. Query processing in a system for distributed databases. Tech. Rep., Computer Corporation of America, Cambridge, Mass.

GOUDA, M., AND DAYAL, U. 1981. Optimal semijoin schedules for query processing in local distributed database systems. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (Ann Arbor, Mich., Apr. 29–May 1). ACM, New York, pp. 164–175.

GRAHAM, M. H. 1979. On the universal relation. Tech. Rep., Dept. of Computer Science, Univ. of Toronto, Toronto, Ont., Canada, Sept.

GRIFFITHS SELINGER, P., et al. 1979. Access path selection in a relational data base management system. Tech. Rep., IBM Research Laboratory, San Jose, Calf., Jan

HEVNER, A. R. 1980. The optimization of query processing on distributed database systems. Ph.D. dissertation, Dept. of Computer Science, Purdue Univ., Lafayette, Ind.

HEVNER, A. R , AND YAO, S. B. 1979. Query processing in distributed database system. *IEEE Trans. Softw. Eng SE-5,* 3 (May), 177–187.

JARKE, M., AND KOCH, J 1983 Range nesting: A fast method to evaluate quantified queries. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (San Jose, Calif., May 23–26). ACM, New York, pp. 196–206.

JARKE, M , AND SCHMIDT, J. 1982. Query processing strategies in the Pascal/R relational database management system. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (Orlando, Fla., June 2–4). ACM, New York, pp 256–264.

KAMBAYASHI, Y. 1981. Compressed semijoins and their applications to distributed query processing. IECE Japan, AL81-54.

KAMBAYASHI, Y., AND YOSHIKAWA, M. 1983 Query processing utilizing dependencies and horizontal decomposition. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (San Jose, Calif., May 23–26). ACM, New York, pp. 55–67.

KAMBAYASHI, Y., YOSHIKAWA, M., AND YAGIMA, S. 1982. Query processing for distributed databases using generalized semijoins. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (Orlando, Fla., June 2–4). ACM, New York, pp. 151–160.

KERSCHBERG, L., TING, P. D., AND YAO, S. B. 1980. Optimal distributed query processing. Bell Laboratories, Holmdel, N. J.

KIM, W. 1982. On optimizing an SQL-like nested query. *Trans Database Syst. 7,* 3 (Sept.), 443–469.

KING, J. J. 1982. QUIST: A system for semantic query optimization in relational databases. In *Proceedings of the 7th International Conference on Very Large Data Bases* (Cannes, Sept. 9–11). IEEE, New York, pp. 510–517.

KRISHNAMURTHY, R., AND MORGAN, S. 1984. Distributed query optimization: An engineering approach. *IEEE Data Eng* 220–227.

KUNG, H. T., AND LEHMAN, P L. 1980. Systolic (VLSI) arrays for relational database operations. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (Santa Monica, Calif., May 14–16). ACM, New York, pp. 105–116.

LUK, W. S., AND BLACK, P. A. 1981. On cost estimation in processing a query in a distributed database system. In *Proceedings of the IEEE 5th International Computer Software and Application Conference* (Chicago, Ill., Nov. 18–20). IEEE, New York, pp. 24–32.

LUK, W. S., AND LUK, L. 1980. Optimal query processing strategies in a distributed database system. Tech. Rep., Dept. of Computer Science, Simon Fraser Univ., Burnaby, B.C., Canada.

REINER, D. 1982 (Guest Ed.). *IEEE Database Engineering Special Issue on Query Processing,* Sept.

ROTHNIE, J. B., AND GOODMAN, N. 1977a. A survey of research and development in distributed database management. In *Proceedings of the 3rd International Conference on Very Large Data Bases* (Tokyo, Oct. 6–8). IEEE, New York, pp. 48–62.

ROTHNIE, J. B., AND GOODMAN, N. 1977b. An overview of the preliminary design of SDD-1· A system for distributed databases. In *Proceedings of the Berkeley Workshop on Distributed Data Management and Computer Networks* (Berkeley, Calif.).

ROTHNIE, J. B., JR., BERNSTEIN, P. A., FOX, S., GOODMAN, N., HAMMER, M., LANDERS, T. A., REEVE, C., SHIPMAN, D. W., AND WONG, E. 1980. Introduction to a system for distributed databases (SDD-1). *ACM Trans Database Syst. 5,* 1 (Mar.), 1–17.

SACCO, G. M. 1984. Distributed query evaluation in local area network. *IEEE Data Eng* 510–516.

STONEBRAKER, M., et al. 1982. Performance analysis of distributed data base systems. *IEEE Database Eng. 5,* 4 (Dec.), 58–65.

ULLMAN, J. D. 1980. *Principle of Database Systems.* Computer Science Press, Rockville, Md.

WAH, B. W., AND LIEN, Y. N. 1984. The file assignment and query processing problems in local multiaccess networks. *IEEE Data Eng* 228–235.

WILLIAMS, R., DANIELS, D., HAAS, L., LAPIS, G., LINDSAY, B., NG, P., OBERMARCK, R., SELINGER, P., WALKER, A., WILMS, P., AND YOST, R. 1981. R*: An overview of the architecture. Tech. Rep., IBM Research Laboratories, San Jose, Calif.

WONG, E. 1977. Retrieving dispersed data from SDD-1: A system for distributed databases. In *Proceedings of the 2nd Berkeley Workshop on Distributed Data Management and Computer Networks* (Berkeley, Calif.), pp. 217–235.

WONG, E. 1981. Dynamic re-materialization: Processing distributed queries using redundant data. In *Proceedings of the Berkeley workshop on Distributed Data Management and Computer Networks* (Berkeley, Calif.).

WONG, E., AND KATZ, R. 1983. Distributing a database for parallelism. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (San Jose, Calif., May 23–26). ACM, New York, pp. 23–29.

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

YAO, S. B. 1977. Aproximating block accesses in database organization. *Commun. ACM 20,* 4 (Apr.), 260–261.

YU, C. T , AND OZSOYOGLU, M. Z. 1979. An algorithm for tree-query membership of a distributed query. In *Proceedings of the IEEE 3rd International Computer Software and Application Conference* (Chicago, Ill., Nov.). IEEE, New York, pp. 306–312.

YU, C. T., LAM, K., AND OZSOYOGLU, M. Z. 1979. Distributed query optimization for tree queries. Dept. of Information Engineering, Univ. of Illinois at Chicago Circle, Oct. Also in *J. Comput Syst. Sci. 29* (1984), 409–445.

YU, C. T., LAM, K., CHANG, C., AND CHANG, S. 1982a Promising approach to distributed query processing. In *Proceedings of the Berkeley Workshop on Distributed Data Management and Computer Networks* (Berkeley, Calif.), pp. 363–390.

YU, C. T., CHANG, C., AND CHANG, Y. 1982b. Two surprising results in processing simple queries in distributed databases. In *Proceedings of the IEEE 6th International Computer Software and Application Conference* (Chicago, Ill., Nov. 8–12). IEEE, New York, pp. 377–384.

YU, C., CHANG, C., TEMPLETON, M., BRILL, D., AND LUND, E. 1983. On the design of a distributed query processing algorithm. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data* (San Jose, Calif., May 23–26). ACM, New York, pp. 30–39.

YU, C., GUH, K., CHANG, C., CHEN, C., TEMPLETON, M., AND BRILL, D. 1984a. Placement dependency and aggregate processing in fragmented distributed database environment. In *Proceedings of the IEEE 8th International Computer Software and Application Conference* (Chicago, Ill.). IEEE, New York.

YU, C., GUH, K., CHANG, C., CHEN, C., TEMPLETON, M., AND BRILL, D. 1984b. An algorithm to process queries in a fast distributed network. In *Proceedings of the IEEE Real Time Systems Symposium* (Austin, Tex.). IEEE, New York, pp. 115–122.