JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 3, 137–157 (1986)

An Optimization of Queries in Distributed Database Systems

CHIN-WAN CHUNG

Computer Science Department, General Motors Research Laboratories, Warren, Michigan 48090

AND

KEKI B. IRANI

Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan 48109

Received January 1, 1984

This paper addresses the processing of a query in distributed database systems using a sequence of semijoins. The objective is to minimize the intersite data traffic incurred by a distributed query. A method is developed which accurately and efficiently estimates the size of an intermediate result of a query. This method provides the basis of the query optimization algorithm. Since the distributed query optimization problem is known to be intractable, a heuristic algorithm is developed to determine a low-cost sequence of semijoins. The cost comparison with an existing algorithm is provided. The complexity of the main features of the algorithm is analytically derived. The scheduling time for sequences of semijoins is measured for example queries using the PASCAL program which implements the algorithm.

1. INTRODUCTION

The concept of distributed database systems has emerged as a natural solution to the information processing problems of geographically dispersed organizations. In this paper, we are concerned with processing a query in a distributed relational database system implemented on a point-to-point packet switching communication network.

In order to process the distributed query, portions of the database at dispersed sites have to be transferred to the user site. In a packet switching network, the delay in transmitting a large amount of data between two sites

0743-7315/86 $3.00

is roughly proportional to the volume of the data transmitted [13]. It was first observed [17] for the Arpanet that the data transfer rate between sites is low. Consequently, the minimization of the intersite data transfer is important in processing a distributed query.

The usual methodology for distributed query processing consists of reducing the referenced relations using a sequence of semijoins after initial local processing (ILP). An estimation of the sizes of the reduced relations after each semijoin in a sequence of semijoins is necessary to compute the intersite data transfer incurred by a distributed query.

A number of methods have been proposed which estimate the sizes of intermediate results of queries in centralized database systems [5, 8, 14, 16]. It is difficult, however, to apply these estimation methods to the distributed query processing. Moreover, there has been very little research on the estimation of the sizes of the reduced relations for a sequence of semijoins.

The previous semijoin strategies for distributed query optimization [3, 4, 7, 10, 11] assume that the joining attributes in referenced relations are independent throughout the processing of a query. An improved estimation method is introduced in [1]. However, this method involves graph search which can be quite costly for large graphs. In Sections 3 and 4, we describe our own model for deriving the estimates. Efficient estimation algorithms and formulas are developed in Section 5 based on the model.

The distributed query optimization problem is known to be NP-hard [10]. Hence any realistic algorithm for determining a sequence of semijoins involves heuristics. Our algorithm is no exception. The algorithms which schedule reasonable semijoin strategies for general distributed queries are reported in [1, 3, 11]. The algorithm in [11] constructs a schedule for each relation separately. The algorithm in [1] is based on hill-climbing technique with two enhancements to the basic algorithm. The algorithm in [3] decomposes a query into simple queries [11] and schedules a sequence of semijoins for each simple query. Hence the reduction in cost achievable by processing one simple query for another simple query has not been taken into account. We present our algorithm in Section 6. Our algorithm makes use of the dependence of joining attributes in the same relations to reduce the query cost.

Comparisons of the query costs produced by our algorithm with those obtained by the algorithm given in [1] have been made using the examples given in [1, 3, 11]. In all cases, our algorithm performs better than the other algorithm.

2. Framework of Model

In this section, we state assumptions and explain notations. Then, a model for distributed query optimization is outlined.

It is assumed that a relation is a unit of distribution. It is also assumed that

queries are conjunctive and contain only equijoin terms. During ILP, all the selections, projections, and local joins are performed.

An attribute A in relations R and S is named $R.A$ and $S.A$, respectively. These are considered to be different attributes, since $R.A$ and $S.A$ represent different sets of values. The set of joining attributes is partitioned into blocks $\{B_k | k = 1, \ldots, p\}$ by the equivalence relation $=$. The attributes in each block have a common domain.

For example, for relations

SUPPLIER (S#, S_NAME), SUPPLY (S#, P#), PART (P#, P_NAME)

and the query

FIND (SUPPLIER.S_NAME, PART.P_NAME)
WHERE (SUPPLIER.S# = SUPPLY.S#) AND (SUPPLY.P# = PART.P#),
B_1 = {SUPPLIER.S#, SUPPLY.S#} and
B_2 = {SUPPLY.P#, PART.P#}.

Two different joining attributes are said to be *associated* with each other if they are the attributes of the same relation.

The semijoin of R by S on A and B, denoted by $R{<}A = B]S$, is defined as $(R[A = B]S)[A_R]$, where A_R denotes the attributes of R. If R and S are at different sites, $R{<}A = B]S$ is called *semijoin from* $S.B$ to $R.A$ because $S[B]$ is transmitted to be compared with $R[A]$. After the semijoin from $S.B$. to $R.A$, R is reduced to $R{<}A = B]S$. A semijoin is possible between any pair of joining attributes in the same block.

We define the following variables:

D_k	the domain of the attributes in block B_k
a_i	the ith joining attribute in a query
K_i	the current set of values of joining attribute a_i
A_i	the initial set of values of a_i after ILP
$\|X\|$	the cardinality of a set X
f_{ij}	the semijoin from a_i to a_j
ϕ_i	the ith semijoin in a sequence of semijoins
c_i	the cost incurred by ϕ_i
b_i	the benefit achieved by ϕ_i
n_i	the net benefit of ϕ_i ($n_i = b_i - c_i$)
R_i	the ith relation referenced by a query
w_i	the width of the attribute a_i
σ	a sequence of semijoins

It is assumed that $|A_i|$ and $|R_j|$ after ILP are known for all A_i and R_j necessary to process the query. Also, $|D_k|$ for all B_k and the widths of all the attributes in a query are available from the data directory.

We express the transmission cost in terms of the volume of data traffic. The

data traffic includes the following message overhead incurred by intersite data flow:

(1) V_f: the fixed portion of the message overhead,
(2) V_p: the portion of the message overhead which is proportional to the length of the message.

The transmission cost to transfer the message of length M bytes is given by

$$\begin{aligned} C(M) &= V_f + V_p + M \\ &= V_f + vM, \end{aligned} \tag{1}$$

where $v = 1 + V_p/M$. V_f and v are the parameters determined by the communication network being used.

Assuming the transmission cost function given by (1), we can compute the cost and the benefit of a semijoin. Let c_{ij} be the cost incurred by f_{ij} and b_{ij} the benefit achieved by f_{ij}. The cost c_{ij} is the transmission cost, as given by (1), to transfer K_i and benefit b_{ij} is the reduction in the transmission cost due to the reduction in relations.

The benefit achieved by a semijoin depends on the type of relations involved. When a relation referenced by a query consists of only one joining attribute after ILP, this relation can be ignored after transmitting a semijoin from this attribute. In addition, if the block containing this attribute has only two attributes and neither is in the target list, then both of them can be ignored after the semijoin. Since this situation can cause a chain effect to other relations, additional reductions of intersite data transfer can be achieved. A relation consisting of only one joining attribute after ILP is called a *singleton joining relation*. The set of all singleton joining relations is denoted by SJR. It is common for a query which does not have a long target list to have SJR $\neq \emptyset$, where $\emptyset$ denotes an empty set.

Consider a semijoin f_{ij}, where: (i) a_i is an attribute of R_p; (ii) a_j, a_k are attributes of R_q; and (iii) a_i, $a_j \in B_h$. The benefit b_{ij} for the semijoin f_{ij} is classified into the following four different parts:

1. b_{ij}^1: the benefit due to the reduction of $|R_q|$ which results from the reduction of $|K_j|$ if R_q is not at the user site,
2. b_{ij}^2: the benefit due to the elimination of R_p if R_p is not at the user site and $R_p \in$ SJR,
3. b_{ij}^3: the benefit due to the elimination of a_j if R_q is not at the user site and $B_h = \emptyset$ after f_{ij},
4. b_{ij}^4: the benefit due to the elimination of duplicated values of a_k if R_q is not at the user site and R_q becomes a singleton joining relation after f_{ij}.

Then $b_{ij} = b_{ij}^1 + b_{ij}^2 + b_{ij}^3 + b_{ij}^4$. The components of b_{ij} are illustrated in Fig. 1. Semijoin f_{ij} is called *beneficial* if $b_{ij} - c_{ij} > 0$.

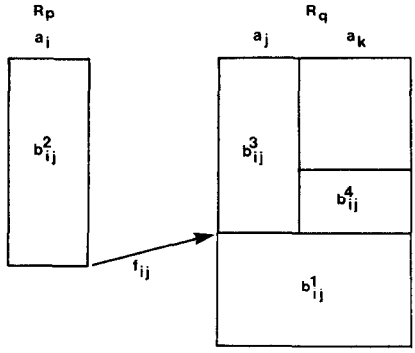

FIG. 1. The components of the benefit b_{ij} achieved by semijoin f_{ij}.

3. ESTIMATION OF THE CARDINALITY OF A RELATION REDUCED BY A SEMIJOIN

Consider a relation R_g and its attribute a_j such that $|K_j| = m$. Define a counting random variable X_i for each $v_i \in K_j$ which counts the number of tuples in R_g in which the value of a_j is v_i. Then for each X_i, the possible values are 1, 2, . . . , $|R_g|-m+1$ and $\Sigma_{i=1}^{m} X_i = |R_g|$. By taking the expected values on both sides of the equation,

$$E\left[\sum_{i=1}^{m} X_i\right] = \sum_{i=1}^{m} E[X_i] = mE[X_i] = |R_g|,$$

where $E[X]$ denotes the expected value of X. Hence

$$E[X_i] = \frac{|R_g|}{|K_j|} \qquad \text{for all } v_i \in K_j. \tag{2}$$

If a semijoin changes the value of a variable, we will append "N" to the name of the variable to designate the new value. After applying a semijoin f_{ij}, R_g and K_j are reduced to $R_g\mathrm{N}$ and $K_j\mathrm{N}$, respectively. Since $|R_g\mathrm{N}| = \Sigma_{v_i \in K_j\mathrm{N}} X_i$, from (2),

$$|R_g\mathrm{N}| = E\left[\sum_{v_i \in K_j\mathrm{N}} X_i\right] = \sum_{v_i \in K_j\mathrm{N}} E[X_i] = |K_j\mathrm{N}|E[X_i] = |K_j\mathrm{N}|\frac{|R_g|}{|K_j|}. \tag{3}$$

Therefore, if we know $|K_j\mathrm{N}|$ for some a_j which is an attribute of R_g, then we can compute $|R_g\mathrm{N}|$.

There are two different ways by which a K_j can be reduced.

(1) If attributes a_i and a_j are in a block, a semijoin f_{ij} reduces K_j to a new set K_iK_j, where K_iK_j denotes $K_i \cap K_j$.

(2) If a_j and a_s are associated with each other, then they are in different blocks. The reduction of K_s by the semijoin f_{rs}, where a_r is in the same block as a_s, also reduces K_j.

First, we discuss the estimation of $|K_j\mathrm{N}|$ due to a semijoin f_{ij}. Suppose $B_k = \{a_1, \ldots, a_i, a_j, \ldots, a_n\}$. Consider f_{ij} as the first element in a sequence of semijoins to process a query. If D_k is perceived to be the sample space, any $X \subset D_k$ is the probabilistic event that $v \in X$ for $v \in D_k$. Initially $K_h = A_h$ for all $a_h \in B_k$, and the only restriction on A_h's is that they be subsets of D_k. Therefore the events A_h's are mutually independent events. After f_{ij}, $K_j\mathrm{N} = A_iA_j$. From $P(A_iA_j) = P(A_i)P(A_j)$,

$$|K_j\mathrm{N}| = |K_i|\frac{|K_j|}{|D_k|}. \tag{4}$$

A dependence between the events K_i and K_j is created as a consequence of the initial f_{ij}. A new procedure must be established to derive the correct estimation of $|K_j\mathrm{N}|$ or $|K_i\mathrm{N}|$ for a subsequent f_{ij} or f_{ji}, respectively. We generalize (4) by using conditional probability.

DEFINITION 1. For a block $B_k = \{a_1, \ldots, a_n\}$, let H be a proper subset of D_k. Consider a sequence of semijoins σ. H is a *reachable set* for B_k after σ if there exists a concatenation V, W of two sequences of semijoins, V and W, where (i) V is a subsequence of σ; (ii) any semijoin in W is between the attributes in B_k; and (iii) the sequence V, W reduces A_i to H for some $a_i \in B_k$.

When a query is partially processed after a sequence σ, the *current set of all reachable sets* for B_k, denoted by RS_k, is defined to be the set containing D_k and all reachable sets for B_k after σ. Since σ is a special case of the sequence V, W, every K_i is an element of RS_k. σ is Λ after ILP, where Λ denotes the null sequence. In this case, $V, W = W$, and the current set of all reachable sets for B_k is called the *initial set of all reachable sets* for B_k, and is denoted by $RS_k^!$.

Consider a set $K \in RS_k$, which is the smallest set containing both K_i and K_j. The knowledge that the event K_j has occurred does not affect the probability of occurrence of the event K_i when the effective sample space is reduced to the event K. Hence the events K_i and K_j are conditionally independent given the event K. That is,

$$P(K_iK_j \mid K) = P(K_i \mid K)P(K_j \mid K). \tag{5}$$

The set K is called the *restricting set* of K_i and K_j. The estimation of $|K_j\mathrm{N}|$ after a semijoin f_{ij} as an element in arbitrary position in a sequence of semijoins is as follows:

$$P(K_iK_j) = P(K_iK_j \mid K)P(K) + P(K_iK_j \mid \bar{K})P(\bar{K}).$$

Since $K_iK_j\overline{K} = \emptyset$ because $K_iK_j \subset K$, we have

$$\frac{|K_iK_j|}{|D_k|} = P(K_iK_j \mid K)\frac{|K|}{|D_k|}. \tag{6}$$

Multiplying both sides by $|D_k|$ and using (5) gives

$$|K_j\mathrm{N}| = |K_i|\frac{|K_j|}{|K|}. \tag{7}$$

To estimate $|K_j\mathrm{N}|$ after a semijoin f_{rs}, where a_r is in the same block as a_s, and a_j and a_s are the attributes in the same relation R_g, a solution to the problem considered by Yao [18] is used. This reduction was ignored in most of the previous semijoin strategies [3, 4, 10, 11]. It was first observed in [7] that Yao's solution is applicable. Suppose $|R_g| = n$, $|K_j| = m$, and $|R_g\mathrm{N}| = k$. Then $|K_j\mathrm{N}|$ after a semijoin f_{rs} is given by

$$m\left[1 - \prod_{i=1}^{k}\frac{n(1 - 1/m) - i + 1}{n - i + 1}\right]. \tag{8}$$

4. Lattice Model for the Current Set of All Reachable Sets

In this section, we show that the current set of all reachable sets RS_k forms a lattice and we subsequently use that fact to calculate the cardinality of the set of values of an attribute. Since the characteristic of a lattice model is common to each block B_k, we will drop the block index unless it is necessary.

4.1. *Initial Lattice*

We begin by showing that the set RS^{I} forms a lattice. This, we show by generating the elements of RS^{I} in a step-by-step manner. The lattice $(RS^{\mathrm{I}}, \subseteq)$ is called an *initial lattice*.

Consider $B = \{a_1, \ldots, a_n\}$. Because of the probabilistic nature of estimation, at any instance during the query processing, $K_iK_j \neq \emptyset$ for any a_i, $a_j \in B$. From the remark after Definition 1 concerning RS^{I}, we observe that for any $X \in RS^{\mathrm{I}}$, X can be reached from A_i for $a_i \in B$ after a sequence of semijoins each of which is between the attributes of B. Hence for any $X \in RS^{\mathrm{I}}$, $X = \bigcap_{i\in J} A_i$, where $J \subseteq \{1, 2, \ldots, n\}$. Let $RS_1^{\mathrm{I}} = \{A_i \mid a_i \in B\}$. The set RS_i^{I} is generated by intersecting i sets taken from RS_1^{I} at a time for $i = 1, 2, \ldots, n$. (See Fig. 2.) Thus

$$RS_2^{\mathrm{I}} = \{A_1A_2, A_1A_3, \ldots, A_{n-1}A_n\}$$
$$\vdots$$
$$RS_n^{\mathrm{I}} = \{A_1A_2 \cdots A_n\}.$$

Let $RS_0^I = \{D\}$. Then $RS^I = \bigcup_{i=0}^{n} RS_i^I$.

The following lemma relates the lattice theory and the query processing in a database system.

LEMMA 1. *Given a block B, the initial set of all reachable sets, RS^I, forms a lattice under set-inclusion, with l.u.b.$\{X, Y\}$ = minimum$\{Z \in RS^I \mid X \subseteq Z$ and $Y \subseteq Z\}$ and g.l.b.$\{X, Y\} = XY$ for any $X, Y \in RS^I$. In estimating the effect of f_{ij} or f_{ji} for $a_i, a_j \in B$, the restricting set is l.u.b.$\{K_i, K_j\}$ and the reduced set is g.l.b.$\{K_i, K_j\}$.*

The first part of Lemma 1 follows directly from the fact that RS^I is closed under intersection [2]. The initial lattice $(RS^I, \subseteq)$ is denoted by L^I.

4.2. *Expanded Lattice*

In this subsection, we show that RS at any instance during the query processing also forms a lattice. The lattice $(RS, \subseteq)$ is called an *expanded lattice*, and is denoted by L.

After ILP, $L = L^I$. The lattice L for a block B can be expanded by semijoins between attributes in other blocks. This is caused by the effect on K_j by a semijoin f_{rs}, where a_r is in the same block as a_s, and a_j is associated with a_s. The new expanded lattice is denoted by $L^N = (RS^N, \subseteq)$. Different expanded lattices will be generated for different sequences of semijoins.

We illustrate this expansion by expanding L^I. Suppose $K_j = X \in L^I$ for $a_j \in B$ before a semijoin f_{rs} is performed. The reduction of K_s results in the reduction of K_j. The reduced K_j, namely K_jN, cannot be expressed by the intersection of sets in RS_1^I. The expansion of an initial lattice that reflects the effect of the semijoin f_{rs} is as follows:

(1) Suppose $RS_1^I = \{A_1, A_2, \ldots, A_n\}$. Let A_{n+1} represent the K_jN formed by f_{rs}.

(2) $RS^N = RS^I \cup \{XA_{n+1}\} \cup G$, where G is the set of elements generated by intersecting XA_{n+1} and the elements in L^I.

Since RS^N contains XA_{n+1} and is closed under intersection, it is the set of all reachable sets for B after f_{rs} and L^N is a lattice. Since $A_{n+1} \subseteq X$, $A_{n+1} = XA_{n+1}$. We prefer to denote the new element by XA_{n+1}. Therefore, A_{n+1} is used when the set is used as a generator while XA_{n+1} is used to designate a specific element in a lattice. For the mth expansion of the lattice, the reduced set will be represented by A_{n+m}.

The expanded lattice is the generalization of the initial lattice L^I. We summarize the above discussion in the following theorem:

THEOREM 1. *Given a block $B = \{a_1, a_2, \ldots, a_n\}$, the current set of all reachable sets RS at any point in the sequence of semijoins forms a lattice under set-inclusion, with l.u.b.$\{X, Y\}$ = minimum$\{Z \in RS \mid X \subseteq Z$ and $Y \subseteq Z\}$ and g.l.b.$\{X, Y\} = XY$ for any $X, Y \in RS$. In estimating the effect*

of f_{ij} or f_{ji} for a_i, $a_j \in B$, the restricting set is l.u.b.$\{K_i, K_j\}$ and the reduced set is g.l.b.$\{K_i, K_j\}$.

The formula to compute $|K_j\mathrm{N}|$ resulting from f_{ij} can be obtained using Theorem 1. Substituting $|K| = |\text{l.u.b.}\{K_i, K_j\}|$ in (7), we have the basic formula

$$\begin{aligned} |K_j\mathrm{N}| &= \text{g.l.b.}|\{K_i, K_j\}| \\ &= \frac{|K_i|\,|K_j|}{|\text{l.u.b.}\{K_i, K_j\}|}. \end{aligned} \tag{9}$$

This lattice model for estimating the reduction of relations during query processing can be used for a broadcasting communication network as well as a point-to-point communication network.

5. Methods for Computing the Cardinality of the Reduced Set

In a distributed query optimization algorithm, the reduction of the set of values of a joining attribute by a semijoin has to be computed frequently. Therefore, an efficient method for estimating the reduciton is crucial in increasing the efficiency of the algorithm.

The special structure and labeling rule of the lattice L are used to indicate lattice operations. In this section, we assume that L contains n initial sets $A_1, \ldots, A_n$ and m sets $A_{n+1}, \ldots, A_{n+m}$ generated by semijoins between the attributes in other blocks. Let I be an index set $\{1, 2, \ldots, n, n+1, \ldots, n+m\}$. Since each reachable set in L is the intersection of some elements of the set $\{A_1, A_2, \ldots, A_n, A_{n+1}, \ldots, A_{n+m}\}$ and since the set RS is closed under intersection we can state the following lemma:

Lemma 2. *Let $I = \{1, 2, \ldots, n, n+1, \ldots, n+m\}$. Any set $X \in RS$ is given by $X = \bigcap_{i \in I_x} A_i$ for some $I_x \subseteq I$.*

I_x is the *index set of the reachable set* X. The index set I_x uniquely determines the set X. The index set of A_{n+k}, $1 \leq k \leq m$, which is obtained after reduction of some $Z \in L$ by a semijoin between the attributes in a block other than B is given, by convention, by $I_{A_{n+k}} = I_z \cup \{n+k\}$.

Lemma 3. *If X and Y are two elements of RS with index sets I_x and I_y, respectively, then g.l.b.$\{X, Y\} = \bigcap_{k \in I_z} A_k$, where $I_z = I_x \cup I_y$ and l.u.b.$\{x, y\} = \bigcap_{h \in I_w} A_h$, where $I_w = I_x \cap I_y$.*

The associative property of g.l.b. allows us to state the following more general result:

Lemma 4. *Let $I = \{1, 2, \ldots, n, n+1, \ldots, n+m\}$. Then for any set $X \in RS$ with the index set $I_x \subseteq I$, $X =$ g.l.b$\{X_1, X_2, \ldots, X_p\}$ for some*

$X_j \in RS, j \in J = \{1, 2, \ldots, p\}$, if and only if $I_x = \bigcup_{j \in J} I_j$, where I_j is the index set of X_j for $j \in J$.

Using Lemma 2, we represent a reachable set by its index set. In computing the cardinality of a set $X \in L$, we determine sets $X_1, X_2, \ldots, X_p$ such that $X_1, X_2, \ldots, X_p \in L$, g.l.b.$\{X_1, X_2, \ldots, X_p\} = X$, and the cardinalities of $X_1, X_2, \ldots, X_p$ are easily obtainable. In fact, the following algorithm generates the index sets of $X_1, \ldots, X_p$ such that either $X_i \in \{A_{n+1}, A_{n+2}, \ldots, A_{n+m}\}$ or $X_i \in L^1$ for $i = 1, \ldots, p$.

```
PROCEDURE SET_COVER (I_x)
  // I_x is the index set of X ∈ RS. I_{A_q} is the //
  // index set of A_q for q = n + 1, ..., n + m //
  C ← ∅              ; initialize the cover being constructed.
  u ← max I_x
  WHILE u > n        ; n: the size of block
  DO BEGIN
    C ← C ∪ {I_{A_u}}
    I_x ← I_x − I_{A_u}
    IF I_x = ∅
    THEN u ← 0
    ELSE u ← max I_x
  END
  IF u ≠ 0 THEN C ← C ∪ {I_x}
  RETURN (C)
END SET_COVER
```

The following is an algorithm which takes the index set of $X \in L$ and computes the cardinality of X by using the associativity of g.l.b. and repetitive application of (9).

```
PROCEDURE CAL_CARD (I_x)
  // I_x is the index set of X. SET_COVER takes I_x and //
  // returns {I_1, ..., I_p} corresponding to {X_1, ..., X_p}//
  IF max I_x ≤ n
  THEN CARD ← |D| Π_{i∈I_x} P(A_i)      ; P(A_i) = |A_i|/|D|
  ELSE BEGIN
    SET_COVER (I_x)
    FOR i = 1 UNTIL p      DO k_i ← max I_i
    IF k_1 ≤ n             ; suppose k_1 ≤ , ..., ≤ k_p
    THEN C_GLB ← |D| Π_{i∈I_1} P(A_i)
    ELSE C_GLB ← |A_{k_1}|
    FOR i = 1 UNTIL p-1
    DO BEGIN
```

```
      LUB ← I_i I_{i+1}   ;LUB is the index set of l.u.b.{X_i, X_{i+1}}
      IF  LUB = ∅
      THEN  C_LUB ← |D|
      ELSE C_LUB ← CAL_CARD (LUB)
      C_GLB ← C_GLB × |A_{k_{i+1}}|/C_LUB   ; use (9)
      I_{i+1} ← I_i ∪ I_{i+1}
    END
    CARD ← C_GLB          ;CARD = |X|
  END
END CAL_CARD
```

In order to use CAL_CARD, we have only to store the information of D, $A_1, \ldots, A_n, A_{n+1}, \ldots, A_{n+m}$ instead of the whole lattice.

Next, we present an approximate formula of (8). Since k is the number of tuples of the reduced relation after a semijoin, k may be very large. In this case, k iteration required in (8) takes a long computation time. We have the following approximate formula:

$$|K_h \mathrm{N}| = \begin{cases} m(1 - (1 - k/n)^{n/m}) & \text{if } n/m < k \\ m(1 - (1 - 1/m)^k) & \text{otherwise.} \end{cases} \tag{10}$$

We have compared the results of (8) and (10) for a broad range of values of n, m, and k. The comparison shows that the error due to the use of (10) instead of (8) is practically negligible.

6. Solution Algorithm

In this section, we present a heuristic algorithm for deriving a sequence of semijoins. Further, we discuss the heuristics on which this algorithm is based.

Our strategy is to process a block as a unit and to sequence the blocks. We select a block, then schedule a sequence of semijoins among the attributes of the block, according to rules which will be explained later. The subsequent blocks are selected and processed in the same way. The blocks are sequenced to take advantage of the reductions in cost by processing a block for other blocks.

Our heuristic algorithm, Algorithm H, consists of the following seven major procedures:

1. INIT_ATTR_INACTIVATION
2. PROCESS_BLOCKS
3. BUILD_PATH
4. REVERSE_PROCESS_BLOCKS
5. HILL_CLIMBING
6. COMPLETION
7. SCREENING

Algorithm H also updates the database state after considering a semijoin, using the estimation method previously presented. The procedures in Algorithm H are sequentially executed in the order as listed above except for procedure BUILD_PATH, which is embedded in procedure PROCESS_BLOCKS.

Procedures PROCESS_BLOCKS, REVERSE_PROCESS_BLOCKS, and COMPLETION are the main features of Algorithm H. Procedures INIT_ATTR_INACTIVATION, BUILD_PATH, HILL_CLIMBING, and SCREENING are control features to increase the robustness of the algorithm for random input data. We shall describe the main features of the algorithm followed by the control features.

6.1. *Main Features*

Procedures PROCESS_BLOCKS selects and processes one block at a time. For an attribute a_i defined on a domain D_k, the *density* d_i is $|K_i|/|D_k|$. Consider a block $B_k = \{a_1, \ldots, a_n\}$ with $d_1 \leq \cdots \leq d_n$. The basic strategy to process a block is to perform the sequence of semijoins, $f_{12}, \ldots, f_{n-1,n}$, to achieve a maximal reduction within a block with a minimal cost. Processing a block in this way is called a *block visit*.

If an attribute a_i in a block B_k is not associated with any attribute in other blocks and if the visit to the block B_k does not end in a_i, then the attribute a_i can be excluded from further scheduling of semijoins because the current values of some other attribute in the block B_k are contained in the values of a_i. After a block is visited, the remaining attributes are called *active attributes*, and the excluded ones *inactive attributes*.

The amount of data transferred by each semijoin in visiting B_k is bounded by $|K_1|w_1$. Since B_k contains at least two joining attributes, d_1d_2 is a rough approximation of the reduction achieved by visiting B_k. We, therefore, multiply $|K_1|w_1$ by a penalty factor $1 + d_1d_2$, and we define the block cost BC(k) of an unvisited block B_k as follows:

$$BC(k) = |K_1|w_1(1 + d_1d_2). \tag{11}$$

The following variables are defined to explain the algorithms:

β	the set of all blocks
Uβ	the set of unvisited blocks
Vβ	the set of visited blocks
SVβ	the sequence of visited blocks
σ_h	the sequence of semijoins being scheduled by Algorithm H
A(B)	the set of active attributes in block B
SI(B)	the sequence of inactive attributes in block B
A$\beta(a_i)$	the set of blocks which contains the attributes associated with a_i. The elements of A$\beta(a_i)$ are called the *associated blocks* of a_i. The set of *end associate blocks*, END_Aβ, at the end of visiting a block B_k which ends in a_i is given by A$\beta(a_i) \cap$ Uβ.

Initially $U\beta = \beta$, $V\beta = \emptyset$, $A(B) = B$, $\sigma_h = \Lambda$, $SI(B) = \Lambda$, and $END\_A\beta = \emptyset$. When a semijoin f_{ij} is scheduled, it is appended to σ_h and if a_i is unassociated, Algorithm H excludes a_i from $A(B)$ and appends it to $SI(B)$. Algorithm H does not append a semijoin f_{ij} to σ_h if f_{ij} does not reduce $|K_j|$ at least by one.

Procedure BUILD_PATH embedded in procedure PROCESS_BLOCKS will be explained more fully later. It builds a path π_k to an unvisited block B_k which is a sequence of semijoins, each between two attributes of a visited block, such that the last semijoin in the path is to an attribute associated with an attribute in B_k. Procedure PROCESS_BLOCKS is described below:

PROCEDURE PROCESS_BLOCKS

While $U\beta = \emptyset$, do the following:

1. (find the candidate blocks) Let B_e and B_u be the blocks with the minimal block cost in $END\_A\beta$ and $U\beta - END\_A\beta$, respectively.

Case 1. If $END\_A\beta = \emptyset$, B_u is the only candidate block.

Case 2. If $END\_A\beta \neq \emptyset$ and $BC(e) \leq BC(u)$, B_e is the only candidate block.

Case 3. IF $END\_A\beta \neq \emptyset$ and $BC(e) > BC(u)$, B_e and B_u are both candidate blocks.

2. (select a block) Let B_{ci} be the ith candidate block for $i = 1, 2$, and B_N the next block to be visited. For each candidate block, call BUILD_PATH. Suppose $\pi_{c1} = \phi_{11}, \ldots, \phi_{1m}$ and $\pi_{c2} = \phi_{21}, \ldots, \phi_{2n}$. If B_{c1} is the only candidate block or $\Sigma_{j=1}^{m} c_{1j} + BC(c1) \leq \Sigma_{j=1}^{n} c_{2j} + BC(c2)$, then $B_N \leftarrow B_{c1}$, else $B_N \leftarrow B_{c2}$.

3. (process a path and visit a block) Append π_N, the path to B_N, to σ_h and the sequence of visited blocks corresponding to the elements of π_N to $SV\beta$. Sort the attributes of $A(B_N)$ in ascending order of densities. Suppose the sorted list is $a_1, \ldots, a_n$, then append $f_{12}, \ldots, f_{n-1,n}$ to σ_h. Let $A\beta = \bigcup_a A\beta(a)$ for $a \in A(B_N) - \{a_n\}$. If a_n is unassociated and $A\beta \cap (U\beta \cup V\beta) \neq \emptyset$, append f_{nj} to σ_h for a_j selected as follows:

Case 1. If $A\beta \cap U\beta \neq \emptyset$, for each $a_i \in A(B_N) - \{a_n\}$ and for each $B_k \in A\beta(a_i) \cap U\beta$, compute the cost of the block B_k, $BC(k)$, after the semijoin f_{ni} is performed. Select a_j which has the minimal $BC(k)$.

Case 2. If $A\beta \cap U\beta = \emptyset$, select a_j associated with the most recently visited block in $SV\beta$.

If $|A(B_N)| > 1$, include B_N in $V\beta$ and append B_N to $SV\beta$. $U\beta \leftarrow U\beta - \{B_N\}$.

In procedure PROCESS_BLOCKS, the reductive power of a visited block is utilized by subsequent block visits whenever possible. In order to use the

reductive power of a block visited later, roughly the order of visits is reversed with respect to the order of visits by procedure PROCESS_BLOCKS.

PROCEDURE REVERSE_PROCESS_BLOCKS

While $SV\beta \neq \Lambda$, do the following:

Let B be the last block in $SV\beta$ with $A(B) = \{a_1, \ldots, a_n\}$ and $d_n \leq , \ldots , \leq d_1$. If $B \in V\beta$ and $n > 1$, append $f_{n,n-1}, \ldots, f_{21}$ to σ_h and exclude B from $V\beta$. Delete B from $SV\beta$.

Procedure COMPLETION reduces the size of the relations containing inactive attributes and at least one target attribute using the reductive power accumulated in active attributes.

PROCEDURE COMPLETION

For each $B \in \beta$, if $A(B) \neq \emptyset$ and $SI(B) \neq \Lambda$, do the following:

Suppose $A(B) = \{a_1, \ldots, a_{i-1}\}$ and $SI(B) = a_i, a_{i+1}, \ldots, a_n$. Select $a_j \in A(B)$ with the minimal density. $s \leftarrow j$. For $t = i$ to n, if the net benefit $n_{st} > 0$, append f_{st} to σ_h and $s \leftarrow t$.

6.2. *Control Features*

Procedure INT_ATTR_INACTIVATION, which is the first procedure in Algorithm H, excludes unassociated attributes with high initial density to avoid semijoins which are neither beneficial for themselves nor useful for subsequent semijoins.

PROCEDURE INIT_ATTR_INACTIVATION

For each $B \in \beta$, do the following:

For each unassociated attribute $a_i \in B$ with $d_i \geq 0.8$: Suppose a_i is an attribute of relation R. If $R \not\subseteq$ SJR or R is at the user site, exclude a_i from $A(B)$ and append a_i to $SI(B)$. Sort the attributes in $SI(B)$ in ascending order of their densities. If $|A(B)| < 2$, exclude B from $U\beta$.

Procedure BUILD_PATH makes better use of the reductive power accumulated in visited blocks to reduce the cost of visiting an unvisited block. We create a path which is a sequence of semijoins, each from an already visited block.

We define the symbols which are used in procedure BUILD_PATH given below. B_N is a candidate for the next block to be visited. $\pi_N = \phi_{N1}, \ldots, \phi_{Nm}$ is a path constructed at some point during the execution of the procedure, where ϕ_{Ni}, $1 \leq i \leq m$, is a semijoin between two attributes in an already visited block B_{Ni}. $P\beta = \{B_{Ni} \mid 1 \leq i \leq m\}$. For each $a \in B_N$ when $\pi_N = \Lambda$ and for each $a \in B_{N1}$ when $\pi_N \neq \Lambda$, we define $CAND\_P\beta(a) = \{B_k \in A\beta(a) \cap V\beta \mid$ there exists $a_h \in B_k$ which is associated with a and $a_g \in A(B_k)$ such that $d_g < d_h\}$. $\alpha = \{a \in B_N \mid CAND\_P\beta(a) \neq \emptyset\}$. a^* is an element of α with minimal density. $n(\pi)$ = net benefit obtained from a sequence of semijoins π.

PROCEDURE BUILD_PATH (B_N)

Initialize $P\beta \leftarrow \emptyset$ and $\pi_N \leftarrow \Lambda$. If $\alpha \neq \emptyset$, do the following:

Select $a^* \in \alpha$. $a_0 \leftarrow a^*$. $\beta^* \leftarrow \text{CAND\_P}\beta(a_0)$. Repeat the following until $\beta^* = \emptyset$:

1. Select $B_j \in \beta^*$ such that if ϕ_{Bj} is the semijoin from the attribute with the smallest density in B_j to the attribute associated with a_0 then $-n(\phi_{Bj}, \pi_N) + \text{BC}(N)$ is minimum.
2. $P\beta \leftarrow P\beta \cup \{B_j\}$. $\pi_N \leftarrow \phi_{Bj}, \pi_N$.
3. Suppose $\phi_{Bj} = f_{xy}$. $a_0 \leftarrow a_x$. $\beta^* \leftarrow \text{CAND\_P}\beta(a_0) - P\beta$.

Select $\pi_{Nh} = \phi_{Nh}, \phi_{N,h+1}, \ldots, \phi_{Nm} \subseteq \pi_N$ such that $-n(\pi_{Nh}) + \text{BC}(N)$ is minimum. $\pi_N \leftarrow \pi_{Nh}$.

A hill-climbing technique can be adopted before using procedure COMPLETION to further decrease the query cost. Only the semijoins between active attributes need to be considered.

PROCEDURE HILL_CLIMBING

Let $\alpha = \bigcup_k \text{A}(B_k)$, where the union is performed over those k for which $|\text{A}(B_k)| > 1$. TEMP_$\alpha \leftarrow \alpha$. While TEMP_$\alpha \neq \emptyset$, do the following:

Select $a_i \in$ TEMP_α with the minimal d_i. Select $a_j \in$ TEMP_α with the maximal n_{ij}. If $n_{ij} > 0$, then append f_{ij} to σ_h and TEMP_$\alpha \leftarrow \alpha - \{a_i\}$, else TEMP_$\alpha \leftarrow$ TEMP_$\alpha - \{a_i\}$.

Procedure SCREENING deletes obviously unnecessary semijoins scheduled by the procedures previously described.

PROCEDURE SCREENING

Let $\sigma_h = \phi_1, \ldots, \phi_t$ and $\phi_k = f_{ij}$. $1 \leq k \leq t$, and let a_j be an attribute of some relation R and a_i the attribute of some relation R'.

1. For $k = t$ down to 1, if $n_k \leq 0$ and ϕ_m neither comes from nor goes to an attribute of R for all $m > k$, delete ϕ_k.
2. If the query references relations at the user site, do the following: For $k = t$ down to 1, if R is at the user site and ϕ_m does not go to an attribute of R' for all $m > k$, replace ϕ_k with "move R' to the user site."

Since the deletion of ϕ_k from σ_h does not affect the costs and benefits of other semijoins, they do not have to be recomputed.

6.3. *Complexity Analysis of Algorithm H*

The measure of the complexity is the number of sequences of semijoins generated by Algorithm H in the process of constructing the final sequence. The existence of procedure HILL_CLIMBING in Algorithm H makes it difficult to derive a narrow-bound time complexity. If there are s possible semijoins and m relations involved in processing a query, then the worst case complexity of procedure HILL_CLIMBING is $O(s \sum_{i=1}^{m} |R_i|)$. Fortunately,

procedure HILL_CLIMBING is a refinement feature and very seldom utilized. The worst case complexity of procedure BUILD_PATH can be easily shown to be $O(|\beta|^3)$. Just like procedure HILL_CLIMBING, procedure BUILD_PATH is very seldom utilized.

Since most queries are handled only by the main features of Algorithm H, we shall consider the complexity of the main features of Algorithm H.

THEOREM 2. *The worst case complexity of the main features of Algorithm H without procedure BUILD_PATH is* $O(n^*|\beta|)$, where $n^* = \max\{|B| \mid B \in \beta\}$.

Proof. Procedure PROCESS_BLOCKS is the dominant procedure which has two major loops, one embedded within the other. For $B = \{a_1, \ldots, a_n\} \in \beta$, procedure PROCESS_BLOCKS generates one sequence of length λ, for $\lambda = 1, \ldots, n-1$, and maximum $n-1$ sequences of length n. The maximum number of sequences that can be generated by procedure PROCESS_BLOCKS is $2(n-1)$. Since $n \leq n^*$, the worst case complexity of procedure PROCESS_BLOCKS is $O(n^*|\beta|)$. ■

7. QUERY EXAMPLES

The examples from recently published papers [1, 3, 11] are selected for tests. We compute the costs for the same examples by Algorithm H and the SDD-1 algorithm [1] using our estimation method for the cardinalities of reduced relations. In order to avoid repetitive details, only the first example is carried out in detail.

Since a statistical estimation method is used, the costs, benefits, and cardinalities are first computed in real numbers, and then the results are given in integers by rounding the real numbers.

EXAMPLE 1. The example by Hevner and Yao [11] is considered. This example is also used by Cheung [3]. The database has the following four relations each of which is located at a different site:

EMPLOYEE (E#, ENAME, SEX), COURSE(C#, CNAME, LEVEL)
STUDENT_COURSE(E#, C#), TEACHER_COURSE(E#, C#, ROOM).

The relation TEACHER_COURSE is at the user site. The relational form of the query is as follows:

```
FIND (EMPLOYEE.ENAME, COURSE.CNAME)
WHERE (EMPLOYEE.E# = STUDENT_COURSE.E#)
    AND (EMPLOYEE.E# = TEACHER_COURSE.E#)
    AND (TEACHER_COURSE.C# = COURSE.C#)
    AND (COURSE.LEVEL = 'Advanced')
    AND (TEACHER_COURSE.ROOM = '103')
    AND (EMPLOYEE.SEX = 'M')
```

The parameters for the query and the database are defined as follows:

R_1 = COURSE, R_2 = TEACHER_COURSE

R_3 = EMPLOYEE, R_4 = STUDENT_COURSE

a_1 = COURSE.C#, a_2 = TEACHER_COURSE.C#

a_3 = TEACHER_COURSE.E#, a_4 = EMPLOYEE.E#

a_5 = STUDENT_COURSE.E#, a_c = COURSE.CNAME

a_e = EMPLOYEE.ENAME.

After ILP, the reduced query and the given initial database state are shown below.

FIND (a_c, a_e)
WHERE $(a_1 = a_2)$ AND $(a_3 = a_4)$ AND $(a_4 = a_5)$

$|R_1| = 100$, $|R_2| = 300$, $|R_3| = 200$, $|R_4| = 600$

$|A_1| = 100$, $|A_2| = |A_3| = |A_4| = 200$, $|A_5| = 600$

$w_i = 1$ for $i = 1, \ldots, 5$

$w_c = 11$, $w_e = 9$

$|D_1| = 400$, $|D_2| = 1000$.

From the reduced query, we have $B_1 = \{a_1, a_2\}$ and $B_2 = \{a_3, a_4, a_5\}$.

In accordance with Hevner and Yao's example, the communication network parameters, V_f and v, are set to 10 and 1, respectively. Then the initial cost, IC, of moving R_1, R_3, *and* R_4 after ILP to the user site is 3830.

Query Cost by Algorithm H. The lattices L_1 and L_2 for B_1 and B_2, respectively, and the changes of K_i's during the application of σ_h are shown in Fig. 2. The expansions of the initial lattices are shown in dotted lines. The changes of values of database state variables after each semijoin in σ_h are shown in Table I along with b_i, c_i, and n_i for each semijoin.

1. INIT_ATTR_INACTIVATION

Since $d_i < 0.8$ for all a_i, all the attributes are initially active.

2. PROCESS_BLOCKS

(1) Since initially END_A$\beta = \emptyset$, a block to be visited is selected from U$\beta = \{B_1, B_2\}$ with the minimum block cost. Using (11),

$$\text{BC}(1) = |K_1| w_1(1 + d_1 d_2) = 100 \times 1 \times (1 + 0.25 \times 0.5) = 112.5$$

$$\text{BC}(2) = |K_3| w_3(1 + d_3 d_4) = 200 \times 1 \times (1 + 0.2 \times 0.2) = 208.$$

Since BC(1) < BC(2), B_1 is selected for the visit.

(2) Since V$\beta = \emptyset$, no path is built for B_1. By procedure PROCESS_

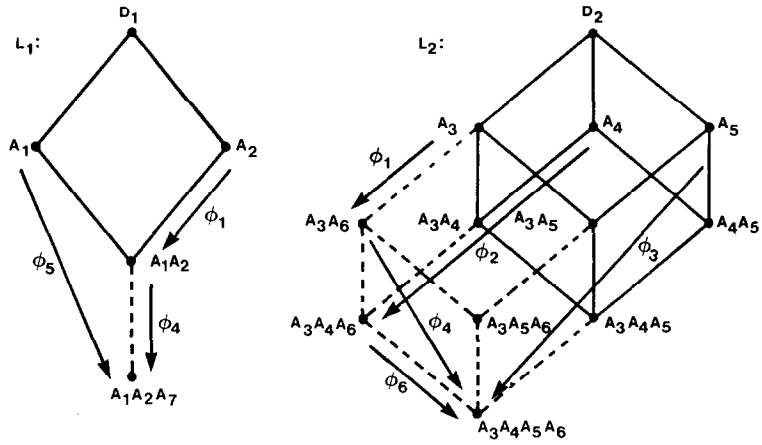

FIG. 2. The expansions of the lattices by the sequence of semijoins generated by Algorithm H for Hevner and Yao's example.

BLOCKS, $\phi_1 = f_{12}$ is appended to σ_h, and a_1 is inactivated. After f_{12}, a new set A_3A_6 which represents the reduced K_3 is formed. After B_1 is visited, $U\beta$ = END_Aβ = $\{B_2\}$. Since B_1 has only one active attribute, B_1 is not included in Vβ.

(3) Since $U\beta = \{B_2\}$, B_2 is the next block to be visited. Since $V\beta = \emptyset$, no path is built for B_2. In procedure BLOCK_VISIT, $\phi_2 = f_{34}$, $\phi_3 = f_{45}$ are appended to σ_h, and a_4 becomes inactive. Vβ becomes $\{B_2\}$ and Uβ becomes $\emptyset$.

3. REVERSE_PROCESS_BLOCKS

Since $V\beta = \{B_2\}$, $\phi_4 = f_{53}$ is appended to σ_h. After f_{53}, a new set $A_1A_2A_7$ which represents the reduced K_2 is formed. Since $R_4 \in$ SJR, R_4 is ignored after f_{53}.

4. HILL_CLIMBING

Since none of the semijoins are beneficial, procedure HILL_CLIMBING does not append any semijoin to σ_h.

5. COMPLETION

(1) For B_1, a_2 is the only active attribute, while a_1 is the only inactive attribute. Hence $\phi_5 = f_{21}$.

(2) For B_2, a_3 is the only active attribute, while a_4 is the only inactive attribute. Hence $\phi_6 = f_{34}$.

6. SCREENING

No semijoin is deleted from σ_h by procedure SCREENING.

From Table I, the cost of the query QC by Algorithm H is

$$\text{QC} = \text{IC} - \sum_{i=1}^{6} n_i = 3830 - 3352 = 478.$$

Query Cost by the SDD-1 Algorithm. We follow the same procedure used

TABLE I
THE SEQUENCE OF SEMIJOINS BY ALGORITHM H AND ITS EFFECT FOR HEVNER AND YAO'S EXAMPLE

i	1	2	3	4	5	6
ϕ_i	f_{12}	f_{34}	f_{45}	f_{53}	f_{21}	f_{34}
$\lvert R_1 \rvert$					8.7	
$\lvert K_1 \rvert$					8.7	
$\lvert R_2 \rvert$	75.0			9.0		
$\lvert K_2 \rvert$	50.0			8.7		
$\lvert K_3 \rvert$	70.0			8.4		
$\lvert R_3 \rvert$		14.0				8.4
$\lvert K_4 \rvert$		14.0				8.4
$\lvert R_4 \rvert$			8.4			
$\lvert K_5 \rvert$			8.4			
b_i	0.0	1860.0	591.6	18.4	1095.3	56.0
c_i	110.0	80.0	24.0	18.4	18.7	18.4
n_i	−110.0	1780.0	567.6	0.0	1076.6	37.6

in SDD-1 and compute the cost of transmitting the data during the sequence of semijoins and transmitting the reduced relations to the assembly site. However, we further include the cost of transmitting the assembled answer from the assembly site to the user site. The cost of the query by the SDD-1 algorithm is 756.

An example by Bernstein *et al.* [1] is considered. In this example, the user site is not specified. We assume that the user site is not one of the sites at which the relations referenced by the user query are located. An example by Cheung [3] is also considered. The results are summarized in Table II. It is shown that Algorithm H performs uniformly better than the SDD-1 algorithm.

For the example in [1], if the relation Y is at the user site, Algorithm H and the SDD-1 algorithm produce identical sequences of semijoins and query cost. However, if the user site is the site of S or P, the query cost according to the SDD-1 algorithm is also 4128. The query cost obtained by Algorithm H is 2611 when S is at the user site whereas 2181 when P is at the user site.

Algorithm H has been implemented in PASCAL and runs on Amdahl

TABLE II
SUMMARY OF QUERY COST COMPARISONS

	Examples by		
Algorithm	H & Y	Bernstein	Cheung
Init. Cost	3,830	206,630	1,330
SDD-1	756	4,128	796
Algorithm H	478	2,711	683

TABLE III
SCHEDULING TIME OF A SEQUENCE OF SEMIJOINS BY ALGORITHM H

Example by	No. of Sites	Scheduling time (Seconds)
Hevner and Yao	4	0.0039
Bernstein *et al.*	3	0.0027
Cheung	4	0.0043

470V/8. We have measured the execution time of the PASCAL program which implements Algorithm H. The scheduling time for the examples using Algorithm H is shown in Table III.

8. CONCLUSION

The query costs obtained by Algorithm H are compared with those obtained by an existing algorithm using some of the published examples. These query costs indicate that Algorithm H is superior. The efficiency of Algorithm H is mainly achieved by the following factors:

(1) Since the number of blocks is considerably less than the number of semijoins, the block-oriented nature of Algorithm H leads to a significant reduction of search space.

(2) The estimation using the lattice model provides an efficient computation method for the dominant term in Algorithm H.

REFERENCES

1. Bernstein, P. A., *et al.* Query processing in a system for distributed databases (SDD-1). *ACM Trans. Database Systems* **6**, 4 (Dec. 1981), 602–625.
2. Birkhoff, G. *Lattice Theory*, 3rd ed. Amer. Math. Soc., Providence, R.I., 1967.
3. Cheung, T. A method for equijoin queries in distributed relational databases. *IEEE Trans. Comput.* (Aug. 1982), 746–751.
4. Chiu, D. M., and Ho, Y. C. A methodology for interpreting tree queries into optimal semi-join expressions. *Proc. 1980 ACM SIGMOD Conference*, May 1980, pp. 169–178.
5. Christodoulakis, S. Estimating selectivities in data bases. Ph.D. dissertation, University of Toronto, 1980.
6. Chung, C. W. A query optimization in distributed database systems. Ph.D. dissertation, University of Michigan, 1983.
7. Computer Corporation of America. A distributed database management system for command and control applications: Final technical report—Part II. Tech. Rep. No. CCA-80–04, Jan 1980.
8. Demolombe, E. Estimation of the number of tuples satisfying a query expressed in predicate calculus language. *Proc. 1980 VLDB Conference*, pp. 55–63.
9. Epstein, R., Stonebraker, M., and Wong, E. Distributed query processing in a relational database system. *Proc. 1978 ACM SIGMOD Conference*, June 1978, pp. 169–180.

10. Hevner, A. R. The optimization of query processing on distributed database systems. Ph.D. dissertation, Purdue University, 1979.
11. Hevner, A. R., and Yao, S. B. Query processing in distributed database systems. *IEEE Trans. Software Engrg.* **SE-5**, 3 (May 1979), 177–187.
12. Irani, K. B., and Khabbaz, N. G. A methodology for the design of communication networks and the distribution of data in distributed supercomputer systems. *IEEE Trans. Comput.* **C-31**, 5 (May 1982), 419–434.
13. Kleinrock, L. *Queueing Systems*, Vol. II, *Computer Applications*. Wiley, New York, 1976.
14. Richard, P. Evaluation of the size of a query expressed in relational algebra. *Proc. 1981 ACM SIGMOD Conference*, pp. 155–163.
15. Rothnie, J. B., *et al.* Introduction to a system for distributed databases (SDD-1). *ACM Trans. Database Systems* **5**, 1 (Mar. 1980), 1–17.
16. Selinger, P., *et al.* Access path selection in a relational database management system. *Proc. 1979 ACM SIGMOD Conference*, May 1979, pp. 23–34.
17. Wong, E. Retrieving dispersed data from SDD-1: A system for distributed databases. *Proc. 1977 Berkeley Workshop on Distributed Data Management and Computer Networks*, May 1977, pp. 217–235.
18. Yao, S. B. Approximating block accesses in database organizations. *Comm. ACM.* **20**, 4 (April 1977), 260–261.