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
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
\[
\text{SUPPLIER (S\#, S\_NAME), SUPPLY (S\#, P\#), PART (P\#, P\_NAME)}
\]
and the query
\[
\text{FIND (SUPPLIER.S\_NAME, PART.P\_NAME)}
\]
WHERE (SUPPLIER.S\# = SUPPLY.S\#) AND (SUPPLY.P\# = PART.P\#),
\]
$B_1 = \{\text{SUPPLIER.S\#, SUPPLY.S\#}\}$ and $B_2 = \{\text{SUPPLY.P\#, PART.P\#}\}$.

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 $R \langle A = B \rangle S$, 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, $R \langle A = B \rangle S$ 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 $R \langle A = B \rangle S$. 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 $i$th 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 $a_j$
- $\phi_i$: the $i$th semijoin in a sequence of semijoins
- $c_i$: the cost incurred by $\phi_i$
- $b_i$: the benefit achieved by $\phi_i$
- $n_i$: the net benefit of $\phi_i$ ($n_i = b_i - c_i$)
- $R_i$: the $i$th relation referenced by a query
- $w_i$: the width of the attribute $a_i$
- $\sigma$: 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

$$C(M) = V_f + V_p + M = V_f + vM,$$  \hspace{1cm} (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 $b_{ij} - c_{ij} > 0$. 
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 $1, 2, \ldots, |R_g| - m + 1$ and $\sum_{i=1}^{m} X_i = |R_g|$. 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

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

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 $R_gN$ and $K_jN$, respectively. Since $|R_gN| = \sum_{v_i \in K_jN} X_i$, from (2),

$$|R_gN| = E\left[ \sum_{v_i \in K_jN} X_i \right] = \sum_{v_i \in K_jN} E[X_i] = |K_jN| E[X_i] = |K_jN| \left| \frac{R_g}{|K_j|} \right|. \quad (3)$$

Therefore, if we know $|K_jN|$ for some $a_j$ which is an attribute of $R_g$, then we can compute $|R_gN|$.

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 $K_i \cap K_j$, where $K_i \cap K_j$ denotes $K_i \cap K_j$. 
(2) If \( a_j \) and \( a_e \) are associated with each other, then they are in different blocks. The reduction of \( K_i \) by the semijoin \( f_{ij} \), where \( a_i \) is in the same block as \( a_i \), also reduces \( K_j \).

First, we discuss the estimation of \( |K_j N| \) due to a semijoin \( f_{ij} \). Suppose \( B_k = \{a_i, \ldots, a_i, a_j, \ldots, a_e\} \). 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 \( f_{ij} \), \( K_j N = A_i A_j \). From \( P(A_i A_j) = P(A_i)P(A_j) \),

\[
|K_j N| = \left| K_i \right| \frac{|K_j|}{|D_k|}. \tag{4}
\]

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 \( |K_j N| \) or \( |K_i N| \) 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 \( \sigma. H \) is a reachable set for \( B_k \) after \( \sigma \) if there exists a concatenation \( V, W \) of two sequences of semijoins, \( V \) and \( W \), where (i) \( V \) is a subsequence of \( \sigma; \) (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 \( \sigma \), 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 \( \sigma \). Since \( \sigma \) is a special case of the sequence \( V, W \), every \( K_i \) is an element of \( RS_k \). \( \sigma \) is \( \Lambda \) after ILP, where \( \Lambda \) 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 \( RS_k^\Lambda \).

Consider a set \( K \in RS_k \), which is the smallest set containing both \( K_i \) and \( K_j \). The knowledge that the event \( K \) 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,

\[
P(K_i K_j | K) = P(K_i | K)P(K_j | K). \tag{5}
\]

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

\[
P(K_i K_j) = P(K_i K_j | K)P(K) + P(K_i K_j | \bar{K})P(\bar{K}).
\]
Since \( K_i K_j \bar{K} = \emptyset \) because \( K_i K_j \subset K \), we have

\[
\frac{|K_i K_j|}{|D_k|} = P(K_i K_j | K) \frac{|K|}{|D_k|}. \tag{6}
\]

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

\[
|K_j N| = |K_i| \frac{|K_j|}{|K|}. \tag{7}
\]

To estimate \( |K_j N| \) after a semijoin \( f_{rs} \), where \( a_i \) 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 \( |R_g N| = k \). Then \( |K_j N| \) 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 \( B_g \), 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 \( (RS^I, \subseteq) \) is called an initial lattice.

Consider \( R = \{a_1, \ldots, a_n\} \). Because of the probabilistic nature of estimation, at any instance during the query processing, \( K_i K_j \neq \emptyset \) for any \( a_i, a_j \in B \). From the remark after Definition 1 concerning \( RS^I \), we observe that for any \( X \in RS^I, X \) 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 \( X \in RS^I, X = \bigcap_{i \in J} A_i \), where \( J \subseteq \{1, 2, \ldots, n\} \). Let \( RS^I_1 = \{A_i | a_i \in B\} \). The set \( RS^I_1 \) is generated by intersecting \( i \) sets taken from \( RS^I \) at a time for \( i = 1, 2, \ldots, n \). (See Fig. 2.) Thus

\[
RS^I_1 = \{A_1 A_2, A_1 A_3, \ldots, A_{n-1} A_n\} \\
\vdots \\
RS^I_n = \{A_1 A_2 \cdots A_n\}.
\]
Let $RS_0^1 = \{D\}$. Then $RS^1 = \bigcup_{i=0}^n RS_i^1$.

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, $RS^1$, forms a lattice under set-inclusion, with $l.u.b\{X, Y\} = \min\{Z \in RS^1 \mid X \subseteq Z \text{ and } Y \subseteq Z\}$ and $g.l.b\{X, Y\} = XY$ for any $X, Y \in RS^1$. 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 $RS^1$ is closed under intersection [2]. The initial lattice $(RS^1, \subseteq)$ is denoted by $L^1$.

### 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, $L = L^1$. 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 $a_i$, and $a_j$ is associated with $a_i$. 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 $L^1$. Suppose $K_j = X \in L^1$ for $a_j \in B$ before a semijoin $f_{rs}$ is performed. The reduction of $K_j$ results in the reduction of $K_j$. The reduced $K_j$, namely $K_jN$, cannot be expressed by the intersection of sets in $RS^1$. The expansion of an initial lattice that reflects the effect of the semijoin $f_{rs}$ is as follows:

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

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

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 $m$th expansion of the lattice, the reduced set will be represented by $A_{n+m}$.

The expanded lattice is the generalization of the initial lattice $L^1$. 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 $l.u.b\{X, Y\} = \min\{Z \in RS \mid X \subseteq Z \text{ and } Y \subseteq Z\}$ 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 $|K_jN|$ resulting from $f_{ij}$ can be obtained using Theorem 1. Substituting $|K| = |l.u.b.\{K_i, K_j\}|$ in (7), we have the basic formula

$$|K_jN| = g.l.b.|\{K_i, K_j\}| = \frac{|K_i| |K_j|}{|l.u.b.\{K_i, K_j\}|}.$$ (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 reduction 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 $I_{A_{n+k}} = I_x \cup \{n+k\}$.

**LEMMA 3.** If $X$ and $Y$ are two elements of $RS$ with index sets $I_x$ and $I_y$, respectively, then $g.l.b.\{X, Y\} = \bigcap_{k \in I_x} A_k$, 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...
\(X_j \in RS, j \in J = \{1, 2, \ldots, p\}, \text{ if and only if } I_x = \bigcup_{j \in J} I_j, \text{ where } I_j \text{ is the index set of } X_j \text{ for } j \in J.\)

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 \(X_1, X_2, \ldots, X_p \in L, \text{ g.l.b.} \{X_1, X_2, \ldots, X_p\} = X, \text{ and the cardinalities of } X_1, X_2, \ldots, X_p \text{ are easily obtainable. In fact, the following algorithm generates the index sets of } X_1, \ldots, X_p \text{ such that either } X_i \in \{A_{n+1}, A_{n+2}, \ldots, A_{n+m}\} \text{ or } X_i \in L' \text{ for } i = 1, \ldots, p.\)

**PROCEDURE SETCOVER \((I_x)\)**

// \(I_x\) is the index set of \(X \in RS. I_{A_q}\) is the // index set of \(A_q\) for \(q = n+1, \ldots, n+m\) //

\[C \leftarrow \emptyset; \text{ initialize the cover being constructed.}\]
\[u \leftarrow \max I_x; \text{ n: the size of block}\]

WHILE \(u > n\) DO BEGIN

\[C \leftarrow C \cup \{I_{A_u}\}\]
\[I_x \leftarrow I_x - I_{A_u}\]
\[\text{IF } I_x = \emptyset \text{ THEN } u \leftarrow 0\]
\[\text{ELSE } u \leftarrow \max I_x\]

END

IF \(u \neq 0\) THEN \(C \leftarrow C \cup \{I_x\}\)

RETURN \((C)\)

**END SETCOVER**

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 \(X \in L. \text{ SETCOVER takes } I_x\) and // returns \\{\(I_1, \ldots, I_p\)\} corresponding to \\{\(X_1, \ldots, X_p\)\}\)

IF \(\max I_x \leq n\) THEN \text{CARD} \leftarrow |D| \prod_{i \in I_x} P(A_i) \quad ; P(A_i) = |A_i|/|D|$

ELSE BEGIN

\text{SETCOVER} \((I_x)\)

FOR \(i = 1 \text{ UNTIL } p\) DO \(k_i \leftarrow \max I_i\)

IF \(k_i \leq n\) THEN \text{C-GLB} \leftarrow |D| \prod_{i = 1} |P(A_i)|

ELSE \text{C-GLB} \leftarrow |A_{k_i}|$

FOR \(i = 1 \text{ UNTIL } p-1\) 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 × $|A_{k+i}|/C_LUB$; use (9)
$I_{i+1} ← I_i \cup I_{i+1}$
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:

$$|K_h N| = \begin{cases} m(1 - (1 - k/n)^{n/m}) & \text{if } n/m < k \\ m(1 - (1 - 1/m)^k) & \text{otherwise.} \end{cases}$$  \hspace{1cm} (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, \( f_1, f_2, \ldots, f_{n-1}, f_n \), 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 \( |K_i|w_1 \). 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 \( |K_i|w_1 \) by a penalty factor \( 1 + d_1d_2 \), and we define the block cost \( BC(k) \) of an unvisited block \( B_k \) as follows:

\[
BC(k) = |K_i|w_1(1 + d_1d_2) .
\]

The following variables are defined to explain the algorithms:

- \( \beta \) the set of all blocks
- \( U\beta \) the set of unvisited blocks
- \( V\beta \) the set of visited blocks
- \( SV\beta \) the sequence of visited blocks
- \( \sigma_s \) the sequence of semijoins being scheduled by Algorithm H
- \( A(B) \) the set of active attributes in block \( B \)
- \( S(I(B)) \) 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, \( END_A\beta \), at the end of visiting a block \( B_k \) which ends in \( a_i \) is given by \( A\beta(a_i) \cap U\beta \).
Initially \( U\beta = \beta, \ V\beta = \emptyset, \ A(B) = B, \ \alpha_h = \Lambda, \ \text{SI}(B) = \Lambda, \) and \( \text{END}_\Delta \text{A} \beta = \emptyset. \) When a semijoin \( f_{ij} \) is scheduled, it is appended to \( \sigma_h \) and if \( a_i \) is unassociated, Algorithm H excludes \( a_i \) from \( A(B) \) and appends it to \( \text{SI}(B). \) Algorithm H does not append a semijoin \( f_{ij} \) to \( \sigma_h \) if \( f_{ij} \) does not reduce \( K_j \) at least by one.

Procedure \text{BUILD\_PATH} embedded in procedure \text{PROCESS\_BLOCKS} will be explained more fully later. It builds a path \( \pi_k \) 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 \text{PROCESS\_BLOCKS} is described below:

**PROCEDURE PROCESS\_BLOCKS**

While \( U\beta = \emptyset, \) do the following:

1. (find the candidate blocks) Let \( B_c \) and \( B_u \) be the blocks with the minimal block cost in \( \text{END}\_\Delta A\beta \) and \( U\beta - \text{END}\_\Delta A\beta, \) respectively.

   *Case 1.* If \( \text{END}_\Delta \text{A} \beta = \emptyset, \) \( B_c \) is the only candidate block.

   *Case 2.* If \( \text{END}_\Delta \text{A} \beta \neq \emptyset \) and \( \text{BC}(e) \leq \text{BC}(u), \) \( B_c \) is the only candidate block.

   *Case 3.* If \( \text{END}_\Delta \text{A} \beta \neq \emptyset \) and \( \text{BC}(e) > \text{BC}(u), \) \( B_c \) and \( B_u \) are both candidate blocks.

2. (select a block) Let \( B_{ci} \) be the \( i \)th candidate block for \( i = 1, 2, \) and \( B_N \) the next block to be visited. For each candidate block, call \text{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 \( \sum_{j=1}^{c1} c_{1j} + \text{BC}(c1) \leq \sum_{j=1}^{c2} c_{2j} + \text{BC}(c2), \) then \( B_N \leftarrow B_{c1}, \) else \( B_N \leftarrow B_{c2}. \)

3. (process a path and visit a block) Append \( \pi_N, \) the path to \( B_N, \) to \( \sigma_h \) and the sequence of visited blocks corresponding to the elements of \( \pi_N \) to \( \text{SV}\beta. \) 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 \( \sigma_h. \) 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 \( f_{n,j} \) to \( \sigma_h \) 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 \( B_k, \) \( \text{BC}(k), \) after the semijoin \( f_{ni} \) is performed. Select \( a_j \) which has the minimal \( \text{BC}(k). \)

   *Case 2.* If \( A\beta \cap U\beta = \emptyset, \) select \( a_j \) associated with the most recently visited block in \( \text{SV}\beta. \)

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

In procedure \text{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 SVβ ≠ Λ, do the following:
Let B be the last block in SVβ with A(B) = \{a_1, \ldots, a_n\} and
d_1 ≤ \ldots ≤ d_n. If B ∈ Vβ and n > 1, append f_{n,n-1}, \ldots, f_{21} to σ_h
and exclude B from Vβ. Delete B from SVβ.

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 ∈ β, if A(B) ≠ ∅ and SI(B) ≠ Λ, do the following:
Suppose A(B) = \{a_1, \ldots, a_i\} and SI(B) = a_i, a_{i+1}, \ldots, a_n. Select
a_j ∈ A(B) with the minimal density. s ← j. For t = i to n, if the net benefit
n_{st} > 0, append f_{st} to σ_h and s ← 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 INT_ATTR_INACTIVATION
For each B ∈ β, do the following:
For each unassociated attribute a_i ∈ B with d_i ≥ 0.8: Suppose a_i is an
attribute of relation R. If R ∉ 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 Uβ.

Procedure BUILD_PATH makes better use of the reductive power accu-
mulated 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. π_N = φ_{N1}, \ldots, φ_{Nm}
is a path constructed at some point during the execution of the procedure,
where φ_{N1}, 1 ≤ i ≤ m, is a semijoin between two attributes in an already
visited block B_{Ni}. Pβ = \{B_{Ni} | 1 ≤ i ≤ m\}. For each a ∈ B_N when π_N = Λ
and for each a ∈ B_{N1} when π_N ≠ Λ, we define CAND_Pβ(a) = \{B_k ∈ Aβ(a) ∩ Vβ | there exists a_h ∈ B_k which is associated with a and
a_x ∈ A(B_k) such that d_x < d_h\}. α = \{a ∈ B_N | CAND_Pβ(a) ≠ ∅\}. a* is
an element of α with minimal density. n(π) = 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 \(a^* \in \alpha\). \(a_0 \leftarrow a^*\). \(\beta^* \leftarrow \text{CAND-P}(a_0)\). Repeat the following until \(\beta^* = \emptyset\):

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

Select \(\pi_{Nh} = \phi_{Nh}, \phi_{Nh+1}, \ldots, \phi_{N_m} \subseteq \pi_N\) such that \(-n(\pi_{Nh}) + BC(N)\) 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 \(\alpha = \bigcup_k A(B_k)\), where the union is performed over those \(k\) for which \(|A(B_k)| > 1\). \(\text{TEMP-} \alpha \leftarrow \alpha\). While \(\text{TEMP-} \alpha \neq \emptyset\), do the following:

Select \(a_i \in \text{TEMP-} \alpha\) with the minimal \(d_i\). Select \(a_j \in \text{TEMP-} \alpha\) with the maximal \(n_{ij}\). If \(n_{ij} > 0\), then append \(f_{ij}\) to \(\sigma_h\) and \(\text{TEMP-} \alpha \leftarrow \alpha - \{a_i\}\), else \(\text{TEMP-} \alpha \leftarrow \text{TEMP-} \alpha - \{a_i\}\).

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 \(\phi_m\) neither comes from nor goes to an attribute of \(R\) for all \(m > k\), delete \(\phi_k\).
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 \(\phi_m\) does not go to an attribute of \(R'\) for all \(m > k\), replace \(\phi_k\) with "move \(R'\) to the user site."

Since the deletion of \(\phi_k\) from \(\sigma_h\) 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 \(O(s \Sigma_{i=1}^{m} |R_i|)\). 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 $\lambda$, 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:

\[
R_1 = \text{COURSE}, \quad R_2 = \text{TEACHER\textunderscore COURSE}
\]

\[
R_3 = \text{EMPLOYEE}, \quad R_4 = \text{STUDENT\textunderscore COURSE}
\]

\[
a_1 = \text{COURSE.C\#}, \quad a_2 = \text{TEACHER\textunderscore COURSE.C\#}
\]

\[
a_3 = \text{TEACHER\textunderscore COURSE.E\#}, \quad a_4 = \text{EMPLOYEE.E\#}
\]

\[
a_5 = \text{STUDENT\textunderscore COURSE.E\#}, \quad a_c = \text{COURSE.CNAME}
\]

\[
a_e = \text{EMPLOYEE.ENAME}
\]

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

\[
\text{WHERE } (a_1 = a_*) \text{ AND } (a_3 = a_4) \text{ AND } (a_4 = a_5)
\]

\[
|R_1| = 100, \quad |R_2| = 300, \quad |R_3| = 200, \quad |R_4| = 600
\]

\[
|A_1| = 100, \quad |A_2| = |A_3| = |A_4| = 200, \quad |A_5| = 600
\]

\[
w_i = 1 \quad \text{for } i = 1, \ldots, 5
\]

\[
w_c = 11, \quad w_e = 9
\]

\[
|D_1| = 400, \quad |D_2| = 1000.
\]

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 \(v_f\), 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 \(\sigma_h\) 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 \(\sigma_h\) 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 \(\text{END\_A} = \emptyset\), a block to be visited is selected from \(U\beta = \{B_1, B_2\}\) with the minimum block cost. Using (11),

   \[
   BC(1) = |K_1|w_1(1 + d_1d_2) = 100 \times 1 \times (1 + 0.25 \times 0.5) = 112.5
   \]

   \[
   BC(2) = |K_3|w_3(1 + d_3d_4) = 200 \times 1 \times (1 + 0.2 \times 0.2) = 208
   \]

   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 \( \sigma_h \), 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, \( U\beta = \text{END} \cdot A\beta = \{B_2\} \). Since \( B_1 \) has only one active attribute, \( B_1 \) is not included in \( V\beta \).

3. REVERSEPROCESS BLOCKS
Since \( V\beta = \{B_2\} \), \( \phi_4 = f_{35} \) is appended to \( \sigma_h \). After \( f_{35} \), a new set \( A_1A_2A_7 \) which represents the reduced \( K_2 \) is formed. Since \( R_4 \in \text{SJR} \), \( R_4 \) is ignored after \( f_{35} \).

4. HILLCLIMBING
Since none of the semijoins are beneficial, procedure HILLCLIMBING does not append any semijoin to \( \sigma_h \).

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 \( \sigma_h \) by procedure SCREENING.

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

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

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

<table>
<thead>
<tr>
<th>$i$</th>
<th>$\phi_i$</th>
<th>$f_{i1}$</th>
<th>$f_{i4}$</th>
<th>$f_{i5}$</th>
<th>$f_{i3}$</th>
<th>$f_{i1}$</th>
<th>$f_{i4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$R_1$</th>
<th>75.0</th>
<th>9.0</th>
<th>8.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_1$</td>
<td>50.0</td>
<td>8.7</td>
<td>8.4</td>
</tr>
<tr>
<td>$K_2$</td>
<td>70.0</td>
<td>8.4</td>
<td>8.4</td>
</tr>
</tbody>
</table>

### Table II

<table>
<thead>
<tr>
<th>Examples by</th>
<th>Algorithm</th>
<th>Bernstein</th>
<th>Cheung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init. Cost</td>
<td>3,830</td>
<td>206,630</td>
<td>1,330</td>
</tr>
<tr>
<td>SDD-1</td>
<td>756</td>
<td>4,128</td>
<td>796</td>
</tr>
<tr>
<td>Algorithm H</td>
<td>478</td>
<td>2,711</td>
<td>683</td>
</tr>
</tbody>
</table>
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


