A Model for Equi-join Query Processing

in Distributed Relational Databases

by

Kuan-Tsae Huang*

Wilbur B. Davenport Jr.

* The authors are with the Dept. of Electrical Engineering and Computer Science and the Laboratory for Information and Decision Systems at Massachusetts Institute of Technology, Cambridge, Mass. 02139.

** This research was conducted at the MIT Laboratory for Information and Decision Systems with support provided by the Office of Naval Research under contract ONR/N00014-77-C-0532.
Abstract

We develop a mathematical model to compute the minimum communication cost of a join-semijoin program for processing a given equi-join query. Some definitions and conditions upon which this paper is based are stated. We define a query processing graph for each equi-join query and characterize the set of join-semijoin programs which solve this query. A rule for estimating the size of the derived relation is derived. The parameters for estimating the size of derived relation form a consistent parameter system. With the assumption of communication cost dominance, the cost functions are linear in the size of data transmission. An optimization problem for distributed query processing is well formulated.
1. Introduction

Query processing in distributed relational databases corresponds to the translation of requests, formulated in a nonprocedural relational calculus like language, into a sequence of relational algebra operations which retrieve and update data stored in the distributed database management systems (DDBMSs).

Given a database schema \( D = \{R_1, R_2, ..., R_n\} \), a query can usually be written in a number of alternative algebraic expressions. In particular, each query can be put in the following form:

\[
Q = \prod_{T_L} \sigma_q (R_1 \times R_2 \times ... \times R_n)
\]

where \( T_L \) contains the attributes in the answer relation; \( q \) is a predicate and each \( R \) is a relation. Usually, \( T_L \) is referred to as the target-list and \( q \) as the qualification of a query. We shall assume all queries are expressed in this canonical form, denoted by \( Q = (q, T_L) \).

In distributed query processing, the execution of a query involves data transmissions which may take significant time in comparison with the subquery and elementary operation execution times. We assume that the data communication costs dominate the local processing costs, so the local processing cost of a query (e.g., costs of selection and projection) are negligible. In this paper, our objective is to minimize the total data transmission cost for processing a query.
For a query \( Q = (q, TL) \), let \( \{ R_1, R_2, \ldots, R_n \} \) be the set of relation schemas referenced by \( q \) and let \( X \) be the set of attributes appearing in \( q \). Before processing the query, we can project each relation \( R_i \) over attributes \( (X \cup TL) \cap R_i \). We then execute those subqueries which reference to only one local relation.

A query \( Q = (q, TL) \) is a conjunctive equi-join query if the qualification \( q \) is a conjunction of equi-join clauses of the form \((R_i. X = R_j. Y)\), where \( X \) and \( Y \) are subsets of attributes of \( R_i \) and \( R_j \) respectively.

In this paper, we restrict our study to a class of equi-join queries. Although it is a subset of complete relational calculus language, it is a rich and large class of queries in practice.

Data transmission is required when two relations that must be joined reside at different sites. To perform the join, one way is to move entire relation from one site to the other. The other way is to replace a join by semijoins and then perform a join. Assume \( R_1 \) and \( R_2 \) at different sites and we want to join \( R_1 \) and \( R_2 \) at the site of \( R_2 \). Using semijoin strategy, one can send the projection of \( R \) on its joining column to \( R_1 \)'s site and perform a semijoin to reduce \( R_1 \) by \( R_2 \) before sending \( R_1 \) to \( R_2 \)'s site. This will be a profitable tactic only whenever the projection of \( R_1 \) on its joining columns smaller than the amount by which \( R_1 \) is reduced by the semijoin.
Prior works in distributed query processing [WONG77, GBWRR80, CHIU79, HY79] were either limited to strategies of performing semijoins first and then joins or without a consistent parameter system to estimate the size of derived relation.

In this paper, we first state some definitions and conditions upon which this paper based. We then define a query processing graph for each equi-join query and derive a theorem about the set of join-semijoin programs which solve this query. Next, we define a rule for estimating the size of derived relation and prove that the parameter system we defined is consistent. With the definition of cost functions, we develop a mathematical model to compute the minimum communication cost of a join-semijoin program for processing a given query.

2. Query Processing Model

A query $Q$ specified by a qualification $q$ over the relations $R_1$, $R_2$, ..., $R_n$, and by a target list $TL$ can be decomposed into a set of operations \{ $p_1$, $p_2$, ..., $p_k$ \} which will produce the answer to the query, where $p_x \in \mathcal{A}$, the set of relational algebra operators. In general, a query can be decomposed into several different executing sequences which will produce the same answer. We call such an executing sequence a strategy. Let $S(Q)$ denote the set of strategies which answer the query $Q$. The goal of the problem is to minimize the overall cost of executing this query $Q$. We can formulate this problem as
MIN \( f(P,D[0]) = \sum_{i=1}^{k} f_{p_i}(D[i]) \) \\
\( P \in S(Q) \) \\
s.t. \( P = p_1 p_2 \ldots p_L \) \\
\( D[i+1] = p_D(D[i]) \) \\
\( D[0] \) is the initial database state

2.1 Definitions and Assumptions

We assume that the distributed database management system DDBMS consists of a collection of interconnected computers \( S_1, S_2 \ldots S_n \) at different sites. Each computer, known as a node in the network, contains a DBMS. Data are logically viewed in the relational model. By the universal relation interpretation [BFMMUY 81], we assume that each site only consists of one relation.

Data transmission in the network is via communication links. We assume that the transmission cost to send one byte of data between any two sites \( i \& j \) is known and equal to \( c \). So the cost function of transmitting data of volume \( V \) between two sites \( i \& j \) is a linear function \( C_{ij}(V) = c_{ij}V \). We assume that all possible subqueries involving data at single site are preprocessed; this we call local processing. The effort of local processing is to reduce the amount of data that needs further processing. After local processing, the following parameters of the query can be defined.
\[ n = \text{number of sites (i.e. relations) in the remaining query} \]
\[ d_i = |R_i|, \text{ number of attributes in site } R_i \]
\[ X_{ij} = R_i \cap R_j, \text{ the set of attributes of joining domains} \]
\[ r_i = \text{number of tuples in relation } R_i \]
\[ w(A) = \text{the width of data item of attribute } A \text{ in relation } R_i \]
\[ s_i = r_i \cdot \sum_{A \in R_i} w(A), \text{ the size of the relation } R_i \]
\[ w(X_{ij}) = \sum_{A \in X_{ij}} w(A) \]

Next, we define several terminologies used in this paper.

Let \((R_i, r_i)\) and \((R_j, r_j)\) be two relation states and \(X \subseteq R_i \cap R_j\).

**Definition:**

The equi-join of \(R_i\) and \(R_j\) on \(X\), denoted by \(R_i \bowtie X R_j\), is the set of tuples \(t \in R_i \cup R_j\) such that \(t[R_i] \in r_i \land t[R_j] \in r_j\).

The semijoin of \(R_i\) and \(R_j\) on \(X\), denoted by \(R_i \mid X R_j\), equals \(R_i \bowtie X R_j[X]\). Equivalently it equals \(\{t_i \mid \exists t_j \in r_j, \forall t_j[X] = t_j[X]\}\).

The natural join of \(R_i\) and \(R_j\) denoted by \(R_i \bowtie X R_j\), is the join of \(R_i\) and \(R_j\) on \(R_i \cap R_j\).

The natural semijoin of \(R_i\) and \(R_j\), denoted by \(R_i \bowtie X R_j\), is the semijoin of \(R_i\) and \(R_j\) on \(R_i \cap R_j\).

Note that the join (resp. semijoin) operator is weaker than the natural-join (resp. natural semi-join) operator in that:
Definition:

A qualification q is called sub-natural iff for each clause
\[ R_{\alpha} \cdot A_{\alpha k} = R_j \cdot A_j l \cdot A_{\alpha k} = A_j l. \]
q is called natural iff the converse holds as well, i.e.,
for all relation schemas \( R \alpha \) and \( R_j \), and for all \( A_k \in R_{\alpha} \cap R_j \),
\[ R_{\alpha} \cdot A_{\alpha k} = R_j \cdot A_j k \] is a clause of q.

Definition:

Given a database schema \( D = \{ R_1, R_2, \ldots, R_n \} \), a query is called
an natural join query (NJQ) (resp. sub-natural join query, SNJQ) iff there exists a natural qualification (resp. sub-natural qualification) for it and \( TL \subseteq U(D) \).

As shown in [BG 81], any query \( Q = (q, TL) \) with an equijoin
qualification \( q \) and a target list \( TL \) can be efficiently
transformed into an equivalent natural join query. Instead of
the class of equijoin queries, EQJ, we shall construct the query
processing model in terms of the class of natural join query, NJQ.

In DDBMS, we define two types of directed operators.

Definition:

1. \( \langle |X| \rangle_{\alpha} \) (or \( R_{\alpha} \langle |X| \rangle_{\alpha} R_j \)) is the distributed natural join
operator which sends \( R_j \) to \( R_{\alpha} \) and performs natural join of
\( R_{\alpha} \) and \( R_j \) at \( R_{\alpha} \) 's site.
2. \(\llcorner X \lrcorner_j (\text{or } R_\llcorner \llcorner X R_j)\) is the directed natural semijoin operator which project \(X = R_\llcorner \land R_j\) over \(R_j\), send the results to \(R_\llcorner\) and perform the join of \(R_\llcorner\) and the result at \(R_\llcorner\)'s site. (i.e. \(R_\llcorner \llcorner X \llcorner \llcorner X R_j\llcorner\) at \(R_\llcorner\)'s site).

Note that \(\llcorner X \lrcorner_j = R_\llcorner \llcorner X \lrcorner R_j\) and \(\llcorner X \lrcorner_j = R_\llcorner X \llcorner R_j\) are similarly defined. One can use them interchangeably. The semijoin operation only reduces the relation state without changing relation schema.

Definition:

A join-semijoin program \(P = p_1 p_2 \ldots p_n\) is a sequence of distributed natural join and distributed natural semijoin operators.

A natural join qualification \(q\) with final node at \(R\) can be done by sending all relations \(R_\llcorner\), \(i \neq 1\), to \(R_1\) and performing \(R_1 \llcorner X \llcorner R_2 \llcorner X \ldots \llcorner X \llcorner R_n\llcorner\) at node \(R_1\). So \(R_2 \llcorner X \llcorner R_1\), \(R_3 \llcorner X \llcorner R_1\), ..., \(R_n \llcorner X \llcorner R_1\) or its permutation are join-semijoin programs of this qualification \(q\).

2.2 Query Processing Graph

We define a processing graph of a qualification over a database schema \(D = \{R_i\}_{i=1}^n\) to be a graph with two type of edges. \(V_\llcorner, \ A_\llcorner, B_\llcorner\). \(V_\llcorner\) is the set of node which is equal to \(D\). \(A_\llcorner\) is a set of semijoin edges which is \(\{a_{i,j} = (R_i, R_j) \in A_\llcorner \mid R_i \land R_j \neq \emptyset \text{ and } R_i \neq R_j\}\). We denote it by \(i \longrightarrow j\) with one arrow on the edge. \(B_\llcorner = V_\llcorner \times V_\llcorner \llcorner = \{b_{i,j} \mid i \neq j\}\) is the set of join edges. We denote it by \(i\)
Note that if $R_{i} \cap R_{j} = \emptyset$, then we cannot perform a semijoin between $R_{i}$ and $R_{j}$. So $a$ is not a semijoin edge. If $R_{i} \subseteq R_{j}$, then $R_{i} = R_{i} \cap R_{j}$. The semijoin of $R_{i} \cap R_{j}$, $R_{i} \times > R_{j}$, is the same as join of $R_{i}$ to $R_{j}$, $R \times R$. This operation is covered by join edge $b_{ij}$.

Example:

$R_{1} = \{ A_{1}, A_{2}, A_{3}, A_{4} \}$
$R_{2} = \{ A_{2}, A_{3}, A_{5}, A_{6} \}$
$R_{3} = \{ A_{5}, A_{7}, A_{8} \}$

The processing graph of the natural join qualification $q$ is:

Without loss of generality, from now on we assume that the final node of a query is node 1.

Definition:

A join-semijoin program $P$ is correct for a natural join qualification $q$ if after executing the program $P$, the final node will have a new relation $R'_{1} = R_{1} \times R_{2} \times ... \times R_{n}$. 
Lemma 1:

Any directed path of edges in $B_3$ from $R_{k_0}$ to $R_{k_1}$, $b_{k_0,k_1}, b_{k_1,k_2}, \ldots, b_{k_{l-1},k_l}$ will form a relation $R_{k_0} \mid X \mid R_{k_1} \mid X \mid \ldots \mid X \mid R_{k_l}$ in node $R_{k_l}$.

Proof: We prove this by induction on the length of the path. If $l=1$, then the path is $b_{k_0,k_1}$. After this operation we will have $R_{k_1} \leftarrow R_{k_0} \mid X \mid R_{k_1}$. By induction assumption on $l=1$, $R_{k_{l-1}} \leftarrow R_{k_0} \mid X \mid R_{k_1} \mid X \mid \ldots \mid X \mid R_{k_{l-1}}$. In the case of $l$, for the first $l-1$ edges of this path, $R_{k_{l-1}} = R_{k_0} \mid X \mid R_{k_1} \mid X \mid \ldots \mid X \mid R_{k_{l-1}}$ by induction assumption. After performing $b_{k_{l-1},k_l}$, we will have $R_{k_1} = R_{k_{l-1}} \mid X \mid R_{k_0} = R_{k_0} \mid X \mid R_{k_1} \mid X \mid \ldots \mid X \mid R_{k_l}$.

Lemma 2:

Any directed spanning tree toward node $1$ of edges in $B_3$ will form $R_1 \mid X \mid R_2 \mid X \mid \ldots \mid X \mid R_n$ at node $1$.

Proof: By lemma 1, any path from $R_k$ to $R_1$ will form a new relation at node $1$ by joining all relations in this path. A directed spanning tree will contain each node $i$ exactly once. If we execute the paths toward node $1$ one by one, we will result the relation $R_1 \mid X \mid R_2 \mid X \mid \ldots \mid X \mid R_n$ at node $1$.

Theorem 1:

Let $Q = (q, TL)$ be a natural join query and $TL = R_1 \cup \ldots \cup R_n$. Let $P$ be a join-semijoin program for $q$, then $P$ is correct iff there exist a subset of the set $\{b_{ij}^j\}$ in $P$ which forms a inversely directed spanning tree toward node $R_1$. 
Proof: IF: Natural semijoin only reduce a relation state without losing any correct data and do not change the relation schema. So after performing the sequence of joins in the directed spanning tree toward $R_i$, we get the $R_i |X_1| \ldots |X|R_n$ at node $R_i$. Any other join operation does not change the state. This implies $P$ is correct.

ONLY IF: For each semijoin operation $a_{ij}$ in $P$, $R_i \cap R_j \neq \emptyset$ and $R_i \leq R_j$. So performing the semijoin operations do not move the full relation state from node $R_i$ to node $R_j$. We still need to perform a join to move the full table of $R_i$ to $R_j$. If there do not exist a subset of $\{b_{ij}\}$ in $P$ which form a directed spanning tree toward node $R_i$, then some information of those nodes which do not have a path toward $R_i$ will lose some information.

From theorem 1, we know the set of correct programs of the NJQ qualification is the set of join-semijoin programs such that there exists a directed spanning tree toward $R$ out of the set of join edges in $P$. We denote the set of correct programs by $\mathcal{P}$. The distributed query processing problem becomes to find a program $p \in \mathcal{P}$ with minimum communication cost. For a program $p$, if we change the order of the sequence of operations, the total communication cost will be different. The set of correct programs is very large. In fact, after executing one operation in $P$, it will change the number of rows and columns of some relations. This change then affects the communication cost of next operation. So the communication cost of one operation will depend
on the previous subsequence of operations.

2.3 Estimate the Size of the Derived Relations

In order to compare the communication cost of query processing strategies, it is very important to have a method to estimate the size of a relation after one operation. Also the system for estimation of the size of the derived relation must be consistent in the sense that if two sequences of operations will produce the same results, the estimated sizes of the result according to the two sequences of operations must be the same.

We introduce the notion of semijoin reducibility and join reducibility of \( R_i \) to \( R_j \), denoted by \( \alpha_{ij} \) and \( \beta_{ij} \) respectively, for each pair of relations \( R_i \) and \( R_j \). Where \( 0 < \alpha_{ij} \leq 1 \) and \( 0 < \beta_{ij} \leq 1 \). The interpretation of the semijoin reducibility \( \alpha_{ij} \) of \( R_i \) to \( R_j \) is that the percentage of rows of \( R_j \) will be reduced after performing the semijoin \( R_i \times_1 R_j \). At stage \( t \), if the number of rows of \( R_j \) is \( r_j[t-1] \) and the semijoin reducibility of \( R_i \) to \( R_j \) is \( \alpha_{ij}[t-1] \), the number of rows of \( R_j \) after performing semijoin \( R_i \times_1 R_j \) will be reduced to \( r_j[t] = r_j[t-1] \times (1-\alpha_j[t-1]) \). Note that the semijoin reducibility of \( R_i \) to \( R_j \) is not equal to the semijoin reducibility of \( R_j \) to \( R_i \) and \( \alpha_{ij}[t] = 0 \) for all \( t \). The interpretation of the join reducibility of \( R_i \) to \( R_j \) is that after performing join \( R_i \mid X \rangle R_j \) at site \( j \), the number of rows of new relation \( R_i \mid X \rangle R_j \) at site \( j \) will be \( r_j[t] = r_j[t-1] \times r_j[t-1] \times (1-\alpha_j[t-1]) \times (1-\beta_j[t-1]) \times (1-\beta_j[t-1]) \). This is because the affect of join \( R_i \mid X \rangle R_j \) is equivalent to perform the semijoins
R\_i \times X! > R\_i and R\_j \times X! > R\_i and then to perform the join of R\_i to R\_j. Both semijoin reducibilities and join reducibility affect the number of rows of the new relation. The join reducibility of R\_j to R\_i is the same as the join reducibility of R\_i to R\_j, i.e. \beta_{ij}[t] = \beta_{ji}[t]. Also \beta_{ij}[t]=0 for all t. For this paper, we assume the set of reducibilities \{\alpha_{ij}, \beta_{ij}\} can be known in advance by some statistical measurement.

Since the number of rows and columns of a relation will be changed after one operation, the reducibilities of this relation with other relations will be changed too. We define how the reducibilities will be changed after one operation. Assume the database state before the operation p\_t be D={R\_1[t-1],...R\_n[t-1]}, the number of rows of each relation R\_k[t-1] be r\_k[t-1], and the semijoin and join reducibilities of R\_k[t-1] to R\_i[t-1] are \alpha_{ik}[t-1] and \beta_{ik}[t-1].

If the operation p\_t at stage t is a\_j, i.e. R\_i \times X! > R\_j, then the database schema will remain the same and only the number of rows of relation R\_j[t] will be changed to equal to r\_j[t-1] * (1-\alpha_{ij}[t-1]) and the number of rows of all other relations will remain the same.

Since this semijoin operation a\_j will reduce the number of rows of R\_j, the semijoin reducibility of R\_j with all other relations R\_k will be reduced too. The semijoin reducibility \alpha_{jk}[t] of R\_k[t] with all other R\_k[t] will be changed as the following rules:

\alpha_{jk}[t] = \begin{cases} 0 & \text{if } k=\bar{i}, k=\bar{j} \\ \alpha_{jk}[t-1] & \text{if } k=\bar{i}, k=i \text{ or } k=\bar{j}, k=j \\ \alpha_{jk}[t-1] + d_{ik}[t-1] - d_{ik}[t-1] & \text{if } k=\bar{i}, k=\bar{j} \\ \alpha_{jk}[t-1] + d_{ik}[t-1] - d_{ik}[t-1] & \text{if } k=\bar{i}, k=\bar{j} \\ \alpha_{jk}[t-1] & \text{otherwise} \end{cases}
The semijoin operation does not affect the join reducibilities. So all join reducibilities at this stage stay the same as last stage, i.e.

\[ \beta_{ij}[t] = \beta_{ij}[t-1] \text{ for all } h \text{ and } k. \]

If the operation \( p \) at stage \( t \) is \( b_{ij} \), i.e. \( R_i \setminus X \bowtie R_j \), then the database schema at node \( j \), \( R_j[t] \) will change to \( R_j[t-1] \cup R_j[t-1] \), and the relation state at site \( j \) will be \( R_j[t] \setminus X \bowtie R_j[t-1] \). The number of rows of \( R_j[t] \), \( r_j[t] \), equal to \( r_j[t-1] \cdot r_j[t-1] \cdot (1 - \alpha_{ij}[t-1]) \cdot (1 - \alpha_{ij}[t-1]) \cdot (1 - \beta_{ij}[t-1]) \). All other relation states will remain the same. Because this is a join operation, the semijoin reducibilities and join reducibilities will be affected.

The semijoin reducibility of \( R_h[t] \) to \( R_k[t] \) will be changed as follows:

\[
\lambda_{h,k}[t] = \begin{cases} 
0 & \text{if } h = i, k = j \\
\lambda_{h,k}[t-1] & \text{if } h = j, k = i \\
\lambda_{h,k}[t-1] + \lambda_{h,k}[t-1] \cdot \alpha_{h,k}[t-1] & \text{if } h = k, k = i, j \\
\lambda_{h,k}[t-1] + \lambda_{h,k}[t-1] - \alpha_{h,k}[t-1] \cdot \lambda_{h,k}[t-1] & \text{if } h = k, k = i, j \\
\lambda_{h,k}[t-1] & \text{otherwise} 
\end{cases}
\]

The join reducibility of \( R_h[t] \) to \( R_k[t] \) will be changed as follows:

\[ \beta_{h,k}[t] = \beta_{h,k}[t-1] + \beta_{h,k}[t-1] - \beta_{h,k}[t-1] \cdot \beta_{h,k}[t-1] \quad \forall h \neq i, j \]

\[ \beta_{h,i}[t] = \left\{ \begin{array}{ll}
1 - \frac{1}{Y_k[t-1]} & \text{if } Y_k[t-1] \neq 0 \\
1 & \text{if } Y_k[t-1] = 0
\end{array} \right. \]
2.4 Consistent Parameter System

We say that a parameter system is consistent if this parameter system will produce the same estimates of the size of the results when the two programs really produce the same results. We now define a parameter system which we shall use to estimate the size of the derived relations has the parameters \{r_\alpha, x_{ij}, \beta_{ij}\}.

Theorem:

The parameter system \{r_\alpha[t], x_{ij}[t], \beta_{ij}[t]\} we defined above is consistent.

Proof: This is because a semijoin operation and join operation can only affect the size of derived relations once and the rules of updating reducibilities have reflecting this fact. So the order of the operations does not affect the estimate of the size of the derived relation.

Example:

Suppose we have three relations \(R_1, R_2\), and \(R_3\), where \(R_\alpha \land R_j \neq \emptyset\) and \(R_\alpha \notin R_j \lor i, j\); and suppose \(x_{ij}[0], \beta_{ij}[0]\) are given. Then the processing graph will be:

![Diagram](image-url)
Let $P_1 = b_3, b_3$, and $P_2 = a_1, a_1, a_1, b_3, b_3$. The two programs will produce the same results $R_1[0] \times X; R_2[0] \times X; R_3[0]$ at site 1. By the rules of estimating the size of the derived relation, the estimate sizes of $R_1[0] \times X; R_2[0] \times X; R_3[0]$ derived by these two programs will be the same. Which is \( \frac{3}{2} \prod_{i=1}^{3} r_x[0] \times \prod_{j=1}^{3} (1 - w_x[0]) \times \prod_{k=1}^{3} (1 - \beta_x[0]) \).

2.5 Problem Formulation

In order to write down the mathematical formulation of distributed query optimization problem, we need to know the cost function of each operation. From our assumption of linear cost function before, we can write down the cost function of this type of operations at stage $t$. The projection of $R_x$ over $R_x \cap R_y$ may result the number of rows of $R_x$ smaller after compressing. Here we ignore this fact. We assume the number of rows after projection on $R_x$ over $R_x \cap R_y$ equal to:

\[ \omega_{x} = \min \{ r_x[t], \prod_{A \in X_{x_j}} |\text{dom}(A)| \} \]

The cost of operation $a_{x_j}$ will be

\[ \text{Cost}(a_{x_j}) = c_{x_j} \times \omega_{x_j} \times \sum_{A \in X_{x_j}} w(A) \]

and the cost of operation $b_{x_j}$ will be

\[ \text{Cost}(b_{x_j}) = c_{x_j} \times r_x[t] \times \sum_{A \in R_x} w(A) \].

Based on the distributed query processing model we developed, the formulation of the distributed query optimization
problem is as follow:

INPUT:
1. a distributed database schema \( D = (R_1[0], \ldots, R_n[0]) \)
2. the width \( w(A) \) of each attribute \( A \) in \( U(D) \)
3. the number of rows \( r_j[0] \) of each relation \( R_j \)
4. the semijoin reducibility \( \alpha_{i,j}[0] \) of each pair of relations \( R_i \cup R_j \) with \( \alpha_{i,i}[0] = 0 \)
5. the join reducibility \( \beta_{i,j}[0] \) of each pair of relations \( R_i \cup R_j \) with \( \beta_{i,i}[0] = 0 \) and \( \beta_{i,j}[0] = \beta_{j,i}[0] \).

OBJECTIVE:
Find an optimal join-semijoin program to solve the natural join program.

Let \( P = p_1, p_2, \ldots, p_k \), then the problem is to minimize \( \sum_{k=1}^{\infty} \text{cost}(p_k) \) according the rules of updating the parameters and cost functions defined above.

3. Conclusion

We have developed a mathematical model for distributed query processing problem for a class of equi-join queries. We also define rules for estimating the size of derived relation. The parameter system based on those rules is consistent. The future research will be to develop algorithms for solving this problem. The reason for difficulty in solving this problem is that computing the cost of one operation depend on the size of derived relation which is the result of previous operations. A special
case of this problem has been shown [HUA81] to have NP-complete complexity. An efficient optimal algorithm for this problem seems unlikely.
REFERENCES

[BFMMUY 81]

[BG 81]

[CHIU 79]

[GBWRR 80]

[HUA 81]
Huang, K. T. Computation Complexity of Distributed Query Processing Problem. (in preparation)

[HY 79]

[WONG 77]