OPTIMIZING STAR QUERIES IN A DISTRIBUTED DATABASE SYSTEM[1]

Arbee L.P. Chen and Victor O.K. Li

Department of Electrical Engineering
University of Southern California
Los Angeles, California 90089-0272

ABSTRACT

The problem of optimal query processing in distributed database systems was shown to be NP-hard. However, for a special type of queries called star queries, we have developed a polynomial optimal algorithm. In an earlier paper, we described an approach to obtain the optimal semi-join program for a star query by gradually reducing the search space to a minimal set S without making any assumptions on the file sizes and the semi-join selectivities. In this paper, by making certain assumptions on the file sizes and the semi-join selectivities, the size of S can be reduced to unity, i.e., given a star query, we can directly generate the optimal program. Our assumption on selectivities is consistent in the sense that we consider the selectivity of a semi-join based on the current database state, i.e., we take into consideration the reduction effects of all prior semi-joins. We have also included an example which compares the performance of existing heuristic algorithms with our proposed optimal algorithm.

[1]This work was supported in part by the National Science Foundation under Contract No. ECS-8204495 and in part by the Joint Services Electronics Program under Contract No. F49620-81-C-0070.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the Tenth International Conference on Very Large Data Bases.

1. INTRODUCTION

A distributed database management system allows datafiles to be distributed and managed on a network of computers. The distribution of the data is transparent to the users who can access the data as if they were located at one site. In reality, to access data distributed in different computer sites, the transmission of data over communication links is needed. Since communication delay is substantial, an efficient query processing mechanism has to be designed. In this paper, we assume the relational data model (see Codd [11]) in studying the query processing problem.

A query consists of two components: the target list and the qualification. The target list contains target attributes that are of interest to the query, i.e. attributes that will appear in the answer. The qualification, for simplicity, is assumed to be a conjunction of selection and equi-join (we shall simply call it join hereafter) clauses which describe the query. A join clause "R_1 joins R_2 on a" is denoted by $R_1 \overset{a}{\leftrightarrow} R_2$, where R_1 and R_2 are relations, and a is the joining attribute. Associated with this join are two semi-joins: R_1 by R_2 on a, and R_2 by R_1 on a, denoted by $R_2 \overset{a}{\rightarrow} R_1$, and $R_1 \overset{a}{\rightarrow} R_2$ respectively. $R_1 \overset{a}{\rightarrow} R_2$ entails shipping $R_1.a$, attribute a of R_1, to the site where R_2 resides and joining $R_1.a$ with R_2. From the query qualification, we can construct a join graph J = (V,E). The nodeset V consists of relations referenced in the query, and an edge (R_i,R_j) belongs to the edgeset E if $R_i \leftrightarrow R_j$ is in the qualification or is an implied join (if joins $Ri \overset{a}{\leftrightarrow} R_j$ and $R_j \overset{a}{\leftrightarrow} R_k$ are in the qualification, then $R_i \overset{a}{\leftrightarrow} R_k$ is an implied join). Under the assumptions that each site contains one relation, that there is only one copy of each relation, and that the cost of local processing is negligible compared to the transmission cost, the query is usually processed (see Bernstein et. al. [2],

Singapore, August, 1984

Black and Luk [4], Apers, Hevner and Yao [1], Yu et. al. [20], Yu and Chang [21]) as follows:

1. Initial local processing: all local operations including selections and projections are processed.

2. Semi-join processing: the only operations left after initial local processing are joins between relations in different sites. A semi-join program is derived from these remaining join operations and executed to reduce the size of joining relations.

3. Final processing: all relations which are needed to calculate the answer of the query are transmitted to a final site for final processing. The final site can be the query requesting site or the site containing the largest relation needed for final processing as suggested by Bernstein et. al. [2].

An optimal semi-join program is one which requires the least total data transmission cost to process the query. It has been shown by Hevner [15], Huang [17] and Yu et. al. [20] that even for some restricted queries, the optimal query processing problem is still NP-hard. Hevner and Yao [16] also developed an optimal algorithm for a special class of queries called simple queries. A simple query is defined as one where, after initial local processing, each relation referenced contains only one attribute, which is the common joining attribute and also the single target attribute. Another optimal algorithm suggested by Chiu, Bernstein and Ho [10] is for processing chain queries, defined as queries having their join graphs configured as a chain, with the relation containing the target attributes at one end. Recently, Sugihara et. al. [19] presented an optimization algorithm which minimizes the total data transmission cost for simple queries in a distributed database system managed on a star network (a star network consists of a single central computer site and several local computer sites connected to the central computer site by communication lines). In this paper, we continue our earlier research presented in Chen and Li [5], [6] to develop an optimal algorithm for another special class of queries, called star queries.

The following is an outline of this paper. In section 2, we summarize the research results described in Chen and Li [6]. Additional assumptions about file sizes and semi-join selectivities are stated in section 3. Section 4 contains the optimal algorithm and its proof of correctness. An example is also included to compare existing heuristic algorithms with the optimal. We conclude our approach in Section 5.

2. SUMMARY OF PREVIOUS RESEARCH RESULTS

In this section, we summarize the results achieved in our earlier research on deriving optimal semi-join programs for star queries (see Chen and Li [6]).

2.1. Execution Graph

Since it is possible to process and move data in parallel in the distributed environment, a semi-join program can either be a serial program which will be executed serially or a non-serial program which contains some parallel processing. We represent semi-join programs by an execution graph described as follows.

An execution graph is a directed acyclic graph[2] whose nodes represent relations (or sites which contain relations), and whose directed edges represent semi-joins. An edge is directed from its predecessor node to its successor node. A node which is not a predecessor node of any edge is called an end node, and if it is not a successor node of any edge, then we call it a start node. A node u is said to be upstream from a node v, and v is said to be downstream from u, if starting from u, one can reach v by following the directed edges. Two nodes are in sequence if one of them is upstream from the other. An edge A is said to be upstream from an edge B, and B is said to be downstream from A, if starting from the successor node of A, we can reach the predecessor node of B by following the directed edges. Two edges are said to be in sequence if one of them is upstream from the other, and they must be executed serially in the order dictated by the directed edges. Edges not in sequence may be executed in parallel. If two or more edges have the same successor node, say v, then only after all of these semi-joins have been executed, can semi-joins having v as a predecessor node be executed. All edges emanating from v can be executed in parallel.

[2]The execution graph must be acyclic since a cycle will correspond to an infinite loop in the associated semi-join program.

Multiple occurrences of a relation or a semi-join may exist in an execution graph. We denote the occurrence of a relation v in an execution graph by v(i), where i is the occurrence number representing the "version" of v after all semi-joins which have v(i) as the successor node have been executed. If v(i) is a start node, then i is designated 0 to specify that v(0) is the unprocessed version of relation v. For $i < j$, each semi-join $x \rightarrow v(i)$ will be executed earlier than each $y \rightarrow v(j)$. We call v(i) a previous occurrence of v(j) and v(j) a next occurrence of v(i) if $i < j$ and no occurrence of v has occurrence number n, where $i < n < j$. When there is no ambiguity as to the representation of the semi-join program, the occurrence numbers can be omitted.

In the rest of this paper, we assume that all occurrences of a relation are in sequence in an execution graph. If this is not true, we can always transform it to another execution graph where this constraint is satisfied without incurring additional cost. There are two cases to be considered in the transformation procedure. Case 1: there exist two occurrences of a relation, which are both start nodes in the execution graph. We can combine these two start nodes to one to represent an equivalent semi-join program which has the same total data transmission cost (see figure 1(a)). Case 2: there exist two occurrences of a relation u, say u(i) and u(j), where u(i) is the previous occurrence of u(j) and they are not in sequence in the execution graph. If we combine u(i) to u(j), then the resultant semi-join program will have no greater total data transmission cost than that of the original semi-join program (see figure 1(b), note that $u(j) \rightarrow v$ will cost less than $u(i) \rightarrow v$).

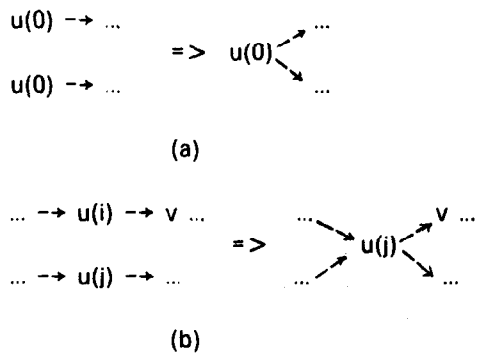


Figure 1: Transformation of Execution Graphs

example 1:

The execution graph representation of the semi-join program

$R_2 \rightarrow R_3$, $R_3 \rightarrow R_2$, $R_2 \rightarrow R_1$, $R_1 \rightarrow R_2$, $R_2 \rightarrow R_3$, $R_3 \rightarrow R_4$

is as follows:

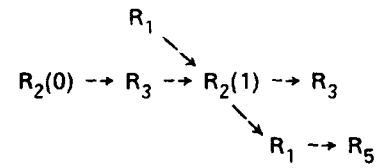
$R_2 \rightarrow R_3 \rightarrow R_2 \rightarrow R_1 \rightarrow R_2 \rightarrow R_3 \rightarrow R_4$

which represents a serial program.

The execution graph representation of the semi-join program

$R_2 \rightarrow R_3$, $R_3 \rightarrow R_2$, $R_1 \rightarrow R_2$, $R_2 \rightarrow R_3$, $R_2 \rightarrow R_1$, $R_1 \rightarrow R_5$

is as follows:



which represents a non-serial program. $R_1 \rightarrow R_2$ and $R_3 \rightarrow R_2$ can be executed in parallel. However, $R_2 \rightarrow R_1$ and $R_2 \rightarrow R_3$ cannot be executed until both of the former have been completed. Note that if we execute $R_1 \rightarrow R_2$ earlier, i.e., delete the edge $R_1 \rightarrow R_2(1)$ and add new edge $R_1 \rightarrow R_2(0)$, then we have the same cost for performing this semi-join. However, since we reduce R_2 earlier, the reduction effect can be achieved and propagated earlier, and the total data transmission cost may be reduced. This technique is called early binding in Luk and Luk [18]. It is also used as one of the enhancement techniques of Algorithm OPT in Bernstein et. al. [2].

Therefore, an execution graph describes the order and the identities of the semi-joins to be executed. Moreover, since each relation is assumed to reside at one site, it also identifies the origin and destination sites of each data transmission. For a semi-join program ρ, we denote its execution graph by $\varepsilon(\rho)$.

2.2. Deriving Optimal Semi-Join Programs for Star Queries

A star query is defined as a query whose join graph is configured as a star with the relation

containing the target attributes as the central node. A general form of star queries expressed in QUEL [14] and its join graph representation are shown in Figure 2.

Retrieve $R_0.t_1$, $R_0.t_2$, ..., $R_0.t_m$

where $R_0.x_1 = R_1.x_1 \wedge R_0.x_2 = R_2.x_2 \wedge \ldots \wedge R_0.x_n = R_n.x_n$

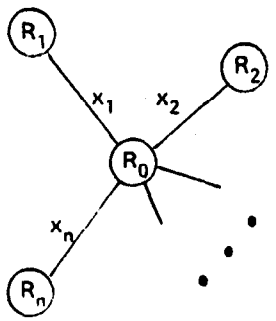


Figure 2: A Star Query and Its Join Graph

A semi-join S is said to be profitable in a semi-join program ρ if, by deleting S from ρ, the total data transmission cost for query processing is increased. A semi-join which is not profitable is said to be unprofitable. Under the assumptions that transferring M units of data between any two sites in the network costs C•M, where C is a constant, and that the query requesting site does not contain any relation referenced in the query, we have shown [6] that each semi-join $R_i \xrightarrow{x_i} R_0$, $i \in \{1, 2, \ldots, n\}$, is always profitable. We call them necessary semi-joins (NSJ's), i.e., they must be included in the optimal program. However, without information on selectivities, the profitability of each $R_0 \xrightarrow{x_i} R_i$ cannot be decided. We therefore call them non-necessary semi-joins (non-NSJ's). Define a semi-join strategy as a semi-join program which includes each semi-join associated with the query exactly once. Also, define an optimal-embedding strategy (OES) as a semi-join strategy from which, by discarding unprofitable non-NSJ's, we can obtain the optimal program. We have shown (again, in [6]) that for a star query, there exists an OES of the form:

$$R_0 \rightarrow R_{i_1} \rightarrow R_0 \rightarrow R_{i_2} \rightarrow R_0 \rightarrow \ldots \rightarrow R_0 \rightarrow R_{i_n} \rightarrow R_0,$$

where $i_j \in \{1, 2, \ldots, n\}$ for $j = 1, 2, \ldots, n$. Without information on selectivities, we cannot decide the profitability of each non-NSJ. That is, each non-NSJ may or may not be included in the optimal program. Therefore, for a star query with n join clauses, there are $n! \cdot 2^n$ candidate optimal programs.

Consider a semi-join strategy of the form

$$R_0 \rightarrow R_{i_1} \rightarrow R_0 \rightarrow \ldots \rightarrow R_{i_n} \rightarrow R_0;$$

If some of the non-NSJ's, say $R_0 \rightarrow R_{i_j}$ is discarded, then the resultant program represented by the execution graph is as follows:

$$R_0 \rightarrow R_{i_1} \rightarrow R_0 \rightarrow \ldots \rightarrow R_0 \rightarrow R_{i_{j-1}} \rightarrow R_{i_j} \rightarrow R_0 \rightarrow R_{i_{j+1}} \rightarrow R_0 \rightarrow \ldots \rightarrow R_{i_n} \rightarrow R_0.$$

By applying early binding, it becomes

$$R_{i_j} \rightarrow R_0 \rightarrow R_{i_1} \rightarrow \ldots \rightarrow R_0 \rightarrow R_{i_{j-1}} \rightarrow R_0 \rightarrow R_{i_{j+1}} \rightarrow \ldots \rightarrow R_{i_n} \rightarrow R_0.$$

Therefore, all $n! \cdot 2^n$ candidate optimal programs will cost no less than the following two forms of programs:

$$(\alpha)\ R_0 \rightarrow R_{i_1} \rightarrow R_0 \rightarrow \ldots \rightarrow R_0 \rightarrow R_{i_n} \rightarrow R_0,$$

$$(\beta)\ \begin{matrix} R_{j_1} \\ \vdots \\ R_{j_k} \end{matrix} \rightarrow R_0 \rightarrow R_{i_{k+1}} \rightarrow R_0 \rightarrow \ldots \rightarrow R_0 \rightarrow R_{i_n} \rightarrow R_0,$$

$1 \leq k \leq n.$

That is, the number of candidate optimal programs is reduced to

$$n! + \sum_{k=1}^{n} \binom{n}{k} \cdot (n-k)!,$$

where the first term is the number of possible programs of form (α) and the second term, that of form (β).

3. ADDITIONAL ASSUMPTIONS

We now make some assumptions about file sizes and semi-join selectivities such that we can continue our approach summarized in section 2 to obtain the optimal program in polynomial time. The optimal algorithm and its correctness will be shown in the next section.

3.1. Assumptions about Relation Sizes and Attribute Sizes

To simplify the optimality problem, every joining attribute, i.e., x_i, $1 \leq i \leq n$, is assumed to have the same "width", say w bytes. Denote the cardinality of a relation R_i by $|R_i|$, and of an attribute $R_i.x_j$ by $|R_i.x_j|$. Since for R_i, $1 \leq i \leq n$, only joining attribute x_i will be left after initial local processing, $|R_i.x_i| = |R_i|$. For R_0, we assume that $|R_0.x_1| = |R_0.x_2| = \ldots = |R_0.x_n| = |R_0|$, i.e., each joining attribute x_i is a candidate key of R_0. The size and the transmission cost from one site to another site of attribute $R_i.x_j$ are therefore $w \cdot |R_i.x_j|$ and $C \cdot w \cdot |R_i.x_j|$ respectively.

3.2. Semi-Join Selectivities

A semi-join $A \xrightarrow{x} B$ selects the tuples of relation B, which have the values of B.x matching those of A.x. Therefore, the selectivity P of $A \dashrightarrow B$ is defined as $P = |B'| / |B|$, where $|B|$ and $|B'|$ are the original and the resultant cardinalities of B respectively. By our assumptions in 3.1, also, $P = |B'.x| / |B.x|$.

Denote $\{A.x\}$ as the set of different values in attribute A.x, $\Pi_{A.x}[A]$ as the projection over A.x on relation A and a||b as the concatenation of two tuples a and b. The following lemma says that if we project over the joining attribute x on relation B after the semi-join $A \xrightarrow{x} B$ is executed, then the resultant set is just the intersection of $\{A.x\}$ and $\{B.x\}$.

Lemma 1: $\Pi_{B.x}[A \xrightarrow{x} B] = \{A.x\} \cap \{B.x\}$.

Proof: $\Pi_{B.x}[A \xrightarrow{x} B] = \Pi_{B.x}[\Pi_B[A \xleftrightarrow{x} B]]$

(by the definition of semi-joins, see Bernstein and Chiu [3])

$= \Pi_{B.x}[\Pi_B[\{a||b: a \in A \wedge b \in B \wedge a.x = b.x\}]]$

(by the definition of joins, see Codd [12])

$= \Pi_{B.x}[\{b: a \in A \wedge b \in B \wedge a.x = b.x\}]$

$= \{b.x: a \in A \wedge b \in B \wedge a.x = b.x\}$

$= \{b.x: a.x \in \{A.x\} \wedge b.x \in \{B.x\} \wedge a.x = b.x\}$

$= \{r: r \in \{A.x\} \wedge r \in \{B.x\}\}$

$= \{A.x\} \cap \{B.x\}$. QED

From forms (α) and (β) in section 2, we can see that each R_i, $1 \leq i \leq n$, has only one occurrence in the execution graph of candidate optimal programs. If it is not a start node, we denote it by $R_i(1)$. Also, we see that there are three types of semi-joins in the execution graph of candidate optimal programs: $R_i(0) \dashrightarrow R_0(a)$, $R_i(1) \dashrightarrow R_0(a)$ and $R_0(b) \dashrightarrow R_i(1)$, where $a > 0$ and $b \geq 0$. Denote X_i as the domain of x_i. We assume that the selectivity of $R_i(0) \dashrightarrow R_0(a)$ in the execution graph of a candidate optimal program, say ρ, is $P_i = |R_i(0)| / |X_i| = |R_i| / |X_i|$ $(0 < P_i \leq 1)$. We further assume that $P_i \leq P_j$ if and only if $|R_i| \leq |R_j|$, $1 \leq i, j \leq n$. The selectivity P_{0i} of $R_0(b) \dashrightarrow R_i(1)$ in ε(ρ) is assumed as $P_{0i} = |R_0(b)| / |X_i|$, $0 < P_{0i} \leq 1$. For each $R_i(1) \dashrightarrow R_0(a)$ in ε(ρ), we have the following lemma for its selectivity P_{i0}.

Lemma 2: For each $R_i(1) \xrightarrow{x_i} R_0(a)$ in ε(ρ), $P_{i0} = P_i$.

Proof: Since $R_i(1)$ is not a start node in ε(ρ), we know that $R_0(a-) \xrightarrow{x_i} R_i(1) \xrightarrow{x_i} R_0(a)$ is in ε(ρ), where $R_0(a-)$ is the previous occurrence of $R_0(a)$ in ε(ρ). By Lemma 1,

$\{R_i(1).x_i\} = \Pi_{R_i(1).x_i}[R_0(a-) \xrightarrow{x_i} R_i(1)]$

$= \{R_0(a-).x_i\} \cap \{R_i.x_i\}$.

Similarly, $\{R_0(a).x_i\} = \Pi_{R_0(a).x_i}[R_i(1) \xrightarrow{x_i} R_0(a)]$

$= \{R_i(1).x_i\} \cap \{R_0(a-).x_i\}$

$= \{R_0(a-).x_i\} \cap \{R_i.x_i\} \cap \{R_0(a-).x_i\}$

$= \{R_0(a-).x_i\} \cap \{R_i.x_i\} = \{R_i(1).x_i\}$.

By the definition of selectivities,

$$P_{i0} = \frac{|R_0(a).x_i|}{|R_0(a-).x_i|} = \frac{|R_i(1).x_i|}{|R_0(a-).x_i|} = \frac{|R_i(1)|}{|R_0(a-)|}$$

$$= \frac{|R_i| \cdot P_{0i}}{|R_0(a-)|} = \frac{|R_i| \cdot \dfrac{|R_0(a-)|}{|X_i|}}{|R_0(a-)|} = \frac{|R_i|}{|X_i|}$$

$= P_i$. QED

3.3. Updating the Cardinality of Relations and Attributes

Lemma 3: After $R_0(a) \xrightarrow{X_i} R_i(1)$ in $\varepsilon(\rho)$ is executed, $|R_i(1)| = |R_0(a)| \cdot P_i$.

Proof: $|R_i(1)| = |R_i| \cdot P_{0i}$

(by the definition of selectivities)

$$= |R_i| \cdot (|R_0(a)| / |X_i|)$$

$$= |R_0(a)| \cdot (|R_i| / |X_i|)$$

$$= |R_0(a)| \cdot P_i. \qquad \text{QED}$$

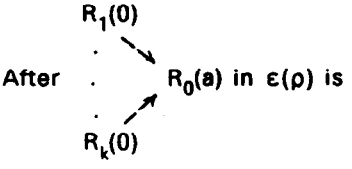
Lemma 4: After $R_0(a-) \rightarrow R_i(1) \rightarrow R_0(a)$ in $\varepsilon(\rho)$ is executed, $|R_i(1)| = |R_0(a)|$.

Proof: $|R_i(1)| = |R_0(a-)| \cdot P_i$ (by Lemma 3)

$$= |R_0(a-)| \cdot P_{i0} \text{ (by Lemma 2)}$$

$$= |R_0(a)|. \qquad \text{QED}$$

Lemma 5:



executed, $|R_0(a)| = |R_0| \cdot \prod_{i=1}^{k} P_i$.

Proof: By the definition of selectivities, the result follows. QED

3.4. The Probability of Non-NSJ's

Each non-NSJ $R_0(a) \rightarrow R_i(1)$ in $\varepsilon(\rho)$, $1 \leq i \leq n$, is followed by an NSJ $R_i \rightarrow R_0$ which must be included in the optimal program. If $|R_0(a)| < |R_i| - |R_i(1)|$, then by deleting $R_0(a) \rightarrow R_i(1)$ from $\varepsilon(\rho)$, the total data transmission cost for ρ is increased. By definition, $R_0 \rightarrow R_i$ is profitable. By Lemma 3, if $|R_0(a)| < |R_i| - |R_i(1)|$, then $|R_0(a)| < |R_i| - |R_0(a)| \cdot P_i$. Therefore, if $|R_0(a)| \cdot (1+P_i) < |R_i|$, then $R_0 \rightarrow R_i$ is profitable. Obviously, when $|R_0(a)| \geq |R_i|$, $R_0 \rightarrow R_i$ is unprofitable.

4. THE OPTIMAL ALGORITHM

In this section, we present an algorithm for generating optimal programs for star queries. A correctness proof of this algorithm is also included.

4.1. The Algorithm OPSTAR

Since each NSJ $R_i \rightarrow R_0$, $1 \leq i \leq n$, is included in any candidate optimal program ρ, R_0 will be fully reduced (see Chiu [9]) after ρ is executed. Since R_0 is the only relation which contains target attributes, once we fully reduce R_0, we also solve the query and obtain the answer. That is, the cost for transmitting the answer to the query requesting site for each candidate optimal program is the same. Therefore, we need only compare the data transmission cost incurred in the execution of candidate optimal programs to obtain the optimal program.

We now present the algorithm OPSTAR to generate the optimal program for star queries. OPT is a queue used to store the optimal program.

Algorithm OPSTAR;

```
begin
  sort and rename R_i, i = 1, 2, ..., n such that
  |R_1| ≤ |R_1| ≤ ... ≤ |R_n|;
  place in OPT a semi-join strategy R_0 → R_1
  → R_0 → R_2 → ... → R_0 → R_n → R_0;
  for i = 1 to n do
    begin
      if |R_0| • ∏_{k=1}^{i-1} P_k • (1+P_i) < |R_i| then exit
      else discard R_0 → R_i from OPT
           (* R_0 → R_i is unprofitable *)
    end
end.
```

The complexity of Algorithm OPSTAR is $O(n \cdot \log_2 n)$.

From the algorithm, it can easily be seen that the optimal program has either of the following two forms:

$$(\gamma) \quad R_0 \rightarrow R_1 \rightarrow R_0 \rightarrow \ldots \rightarrow R_0 \rightarrow R_n \rightarrow R_0,$$

$$(\sigma) \quad \begin{matrix} R_1 \\ \vdots \\ R_{k'} \end{matrix} \quad R_0 \rightarrow R_{k'+1} \rightarrow R_0 \rightarrow \ldots \rightarrow R_0 \rightarrow R_n \rightarrow R_0,$$

$1 \leq k' \leq n$.

example 2:

Given a star query:

retrieve $R_0.t$
where $R_0.x_1 = R_1.x_1 \wedge R_0.x_2 = R_2.x_2 \wedge$
$R_0.x_3 = R_3.x_3 \wedge R_0.x_4 = R_4.x_4$

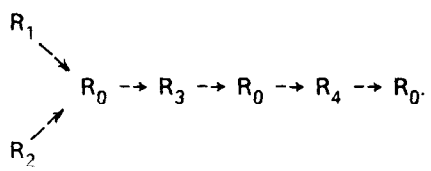
and: $|R_0| = 90$, $|R_1| = 40$, $|R_2| = 50$, $|R_3| = 100$,
$|R_4| = 900$, $|X_1| = 100$, $|X_2| = 100$,
$|X_3| = 125$, $|X_4| = 1000$,

we apply OPSTAR as follows:

1. Since $|R_0|\bullet(1+P_1) = 90\bullet(1+0.4) = 126 > |R_1|$, $R_0 \dashrightarrow R_1$ is unprofitable and discarded.

2. Since $|R_0|\bullet P_1\bullet(1+P_2) = 90\bullet 0.4\bullet(1+0.5) = 54 > |R_2|$, $R_0 \dashrightarrow R_2$ is unprofitable and discarded.

3. Since $|R_0|\bullet P_1\bullet P_2\bullet(1+P_3) = 90\bullet 0.4\bullet 0.5\bullet(1+0.8) = 32.4 < |R_3|$, the algorithm stops and the optimal program is:



4.2. The Correctness of Algorithm OPSTAR

Lemma 6: For $R_0(a-) \dashrightarrow R_i(1) \dashrightarrow R_0(a) \dashrightarrow R_{i+1}(1)$ in $\varepsilon(\rho)$ where $|R_{i+1}| \geq |R_i|$, if $R_0(a-) \dashrightarrow R_i(1)$ is profitable, then $R_0(a) \dashrightarrow R_{i+1}(1)$ is also profitable.

Proof: Since $R_0(a-) \dashrightarrow R_i(1)$ is profitable,

$|R_0(a-)|\bullet(1+P_i) < |R_i|$.

Since $|R_{i+1}| \geq |R_i| > |R_0(a-)|\bullet(1+P_i)$

$= |R_0(a-)| + |R_0(a-)|\bullet P_i$

$\geq |R_0(a-)|\bullet P_i + |R_0(a-)|\bullet P_i\bullet P_{i+1}$

$= (|R_0(a-)|\bullet P_i)\bullet(1+P_{i+1})$

$= |R_i(1)|\bullet(1+P_{i+1})$ (by Lemma 3)

$= |R_0(a)|\bullet(1+P_{i+1})$ (by Lemma 4),

$|R_0(a)|\bullet(1+P_{i+1}) < |R_{i+1}|$. Therefore,

$R_0(a) \dashrightarrow R_{i+1}(1)$ is profitable. QED

Lemma 7: For a semi-join strategy $R_0 \dashrightarrow R_1 \dashrightarrow R_0 \dashrightarrow R_2 \dashrightarrow \ldots \dashrightarrow R_0 \dashrightarrow R_i \dashrightarrow R_0 \dashrightarrow \ldots \dashrightarrow R_0 \dashrightarrow R_n \dashrightarrow R_0$, where $|R_1| \leq |R_2| \leq \ldots \leq |R_n|$, if $R_0 \dashrightarrow R_i$ is a profitable non-NSJ then $R_0 \dashrightarrow R_j$, $i < j \leq n$ are also profitable.

Proof: By inductively applying Lemma 6, the result follows. QED

Theorem 1: Algorithm OPSTAR generates optimal programs.

Proof: We prove the optimality by considering four cases:

$C(\gamma) \leq C(\alpha)$, $C(\gamma) \leq C(\beta)$, $C(\sigma) \leq C(\alpha)$ and $C(\sigma) \leq C(\beta)$,

where C(i) denotes the data transmission cost for executing semi-join programs of form (i). Also, denote $\|R_i\|$ as the cost to transmit the data size $W\bullet|R_i|$ from one site to another site, $0 \leq i \leq n$.

Case 1, $C(\gamma) \leq C(\alpha)$:

By Lemma 3 and Lemma 4, after $R_0(a-) \dashrightarrow R_i(1) \dashrightarrow R_0(a)$ in $\varepsilon(\rho)$ is executed, $|R_0(a)| = |R_i(1)| = |R_0(a-)|\bullet P_i$.

$$C(\gamma) = \|R_0\| + 2\|R_0\|P_1 + 2\|R_0\|P_1P_2 + \ldots + 2\|R_0\|\prod_{i=1}^{n-1} P_i + \|R_0\|\prod_{i=1}^{n} P_i$$

$$C(\alpha) = \|R_0\| + 2\|R_0\|P_{i_1} + 2\|R_0\|P_{i_1}P_{i_2} + \ldots + 2\|R_0\|\prod_{j=1}^{n-1} P_{i_j} + \|R_0\|\prod_{j=1}^{n} P_{i_j}$$

Since by assumption, $P_1 \leq P_2 \leq \ldots \leq P_n$, for each k, $1 \leq k \leq n$, $\prod_{i=1}^{k} P_i \leq \prod_{j=1}^{k} P_{i_j}$.

Therefore, $C(\gamma) \leq C(\alpha)$.

Case 2, $C(\gamma) \leq C(\beta)$:

$$C(\beta) = \prod_{i=1}^{k}\|R_{j_i}\| + \|R_0\|\prod_{i=1}^{k} P_{j_i} + 2\|R_0\|\prod_{i=1}^{k+1} P_{j_i} + \ldots + 2\|R_0\|\prod_{i=1}^{n-1} P_{j_i} + \|R_0\|\prod_{i=1}^{n} P_{j_i}$$

$$C(\gamma) = \|R_0\| + 2\|R_0\|P_1 + \ldots + 2\|R_0\|\prod_{i=1}^{k} P_i + \ldots + 2\|R_0\|\prod_{i=1}^{n-1} P_i + \|R_0\|\prod_{i=1}^{n} P_i$$

$$= \|R_0\| + \|R_0\|P_1 + \|R_0\|P_1 + \ldots + \|R_0\| \prod_{i=1}^{k} P_i + \|R_0\| \prod_{i=1}^{k} P_i + 2\|R_0\| \prod_{i=1}^{k+1} P_i + \ldots + 2\|R_0\| \prod_{i=1}^{n-1} P_i + \|R_0\| \prod_{i=1}^{n} P_i$$

$$= \|R_0\|(1+P_1) + \|R_0\|P_1(1+P_2) + \ldots + \|R_0\| \prod_{i=1}^{k-1} P_i(1+P_k) + \|R_0\| \prod_{i=1}^{k} P_i + \ldots$$

By Lemma 7, every non-NSJ in form (γ) is profitable, we know that $|R_0| \prod_{i=1}^{r-1} P_i(1+P_r) < |R_r|$, $1 \leq r \leq k$. Furthermore, since $|R_1| \leq |R_2| \leq \ldots \leq |R_n|$, $\sum_{i=1}^{k} \|R_i\| \leq \sum_{i=1}^{k} \|R_{j_i}\|$. $\|R_0\|(1+P_1) + \|R_0\|P_1(1+P_2) + \ldots + \|R_0\| \prod_{i=1}^{k-1} P_i(1+P_k) < \sum_{i=1}^{k} \|R_i\| \leq \sum_{i=1}^{k} \|R_{j_i}\|$.

By the similar argument as in case 1,

$\prod_{i=1}^{r} P_i \leq \prod_{i=1}^{r} P_{j_i}$ for $k \leq r \leq n$.

Therefore, $C(\gamma) \leq C(\beta)$.

Case 3, $C(\sigma) \leq C(\alpha)$:

$$C(\sigma) = \sum_{i=1}^{k'} \|R_i\| + \|R_0\| \prod_{i=1}^{k'} P_i + 2\|R_0\| \prod_{i=1}^{k'+1} P_i + \ldots + 2\|R_0\| \prod_{i=1}^{n-1} P_i + \|R_0\| \prod_{i=1}^{n} P_i$$

$$C(\alpha) = \|R_0\| + 2\|R_0\|P_{i_1} + \ldots + 2\|R_0\| \prod_{j=1}^{k'} P_{i_j} + \ldots + 2\|R_0\| \prod_{j=1}^{n-1} P_{i_j} + \|R_0\| \prod_{j=1}^{n} P_{i_j}$$

$$= \|R_0\| + \|R_0\|P_{i_1} + \|R_0\|P_{i_1} + \ldots + \|R_0\| \prod_{j=1}^{k'} P_{i_j} + \|R_0\| \prod_{j=1}^{k'} P_{i_j} + 2\|R_0\| \prod_{j=1}^{k'+1} P_{i_j} + \ldots$$

$$= \|R_0\|(1+P_{i_1}) + \ldots + \|R_0\| \prod_{j=1}^{k'-1} P_{i_j}(1+P_{i_{k'}}) + \|R_0\| \prod_{j=1}^{k'} P_{i_j} + \ldots$$

From form (σ), we know that $R_0 \dashrightarrow R_r$, $1 \leq r \leq k'$ are all unprofitable, i.e., $|R_0| \prod_{i=1}^{r-1} P_i(1+P_r) \geq |R_r|$.

Since $P_1 \leq P_2 \leq \ldots \leq P_n$, $\prod_{i=1}^{r} P_i \leq \prod_{j=1}^{r} P_{i_j}$ for $k' \leq r \leq n$. Also, $|R_r| \leq |R_0| \prod_{i=1}^{r-1} P_i(1+P_r) \leq |R_0| \prod_{j=1}^{r-1} P_{i_j}(1+P_{i_r})$ for $1 \leq r \leq k'$.

Therefore, $\|R_r\| \leq \|R_0\| \prod_{j=1}^{r-1} P_{i_j}(1+P_{i_r})$ for $1 \leq r \leq k'$ and $C(\sigma) \leq C(\alpha)$.

Case 4, $C(\sigma) \leq C(\beta)$:

$$C(\sigma) = \sum_{i=1}^{k'} \|R_i\| + \|R_0\| \prod_{i=1}^{k'} P_i + 2\|R_0\| \prod_{i=1}^{k'+1} P_i + \ldots + 2\|R_0\| \prod_{i=1}^{n-1} P_i + \|R_0\| \prod_{i=1}^{n} P_i$$

$$C(\beta) = \sum_{i=1}^{k} \|R_{j_i}\| + \|R_0\| \prod_{i=1}^{k} P_{j_i} + 2\|R_0\| \prod_{i=1}^{k+1} P_{j_i} + \ldots + 2\|R_0\| \prod_{i=1}^{n-1} P_{j_i} + \|R_0\| \prod_{i=1}^{n} P_{j_i}$$

(i) if $k = k'$, then it is easy to see that $C(\sigma) \leq C(\beta)$.

(ii) if $k' < k$, then

$$C(\sigma) = \sum_{i=1}^{k'} \|R_i\| + \|R_0\| \prod_{i=1}^{k'} P_i + \|R_0\| \prod_{i=1}^{k'+1} P_i + \|R_0\| \prod_{i=1}^{k'+1} P_i + \ldots + \|R_0\| \prod_{i=1}^{k} P_i + \|R_0\| \prod_{i=1}^{k} P_i + 2\|R_0\| \prod_{i=1}^{k+1} P_i + \ldots$$

$$= \sum_{i=1}^{k'} \|R_i\| + \|R_0\| \prod_{i=1}^{k'} P_i(1+P_{k'+1}) + \ldots + \|R_0\| \prod_{i=1}^{k-1} P_i(1+P_k) + \|R_0\| \prod_{i=1}^{k} P_i + \ldots$$

$$< \sum_{i=1}^{k'} \|R_i\| + \sum_{i=k'+1}^{k} \|R_i\| + \|R_0\| \prod_{i=1}^{k} P_i + \ldots$$

(since $R_0 \dashrightarrow R_i$, $k'+1 \leq i \leq k$, are profitable)

$$= \sum_{i=1}^{k} \|R_i\| + \|R_0\| \prod_{i=1}^{k} P_i + \ldots$$

Therefore, $C(\sigma) \leq C(\beta)$.

(iii) if $k < k'$, then

$$C(\beta) = \sum_{i=1}^{k} \|R_{j_i}\| + \|R_0\| \prod_{i=1}^{k} P_{j_i} + \|R_0\| \prod_{i=1}^{k+1} P_{j_i} + \|R_0\| \prod_{i=1}^{k+1} P_{j_i} + \ldots + \|R_0\| \prod_{i=1}^{k'} P_{j_i} + \|R_0\| \prod_{i=1}^{k'} P_{j_i} + 2\|R_0\| \prod_{i=1}^{k'+1} P_{j_i} + \ldots + 2\|R_0\| \prod_{i=1}^{n-1} P_{j_i} + \|R_0\| \prod_{i=1}^{n} P_{j_i}$$

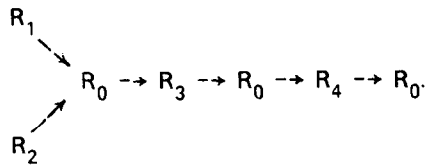
$$= \sum_{i=1}^{k} \|R_{j_i}\| + \|R_0\| \prod_{i=1}^{k} P_{j_i}(1+P_{j_{k+1}}) + \ldots + \|R_0\| \prod_{i=1}^{k'-1} P_{j_i}(1+P_{j_{k'}}) + \|R_0\| \prod_{i=1}^{k'} P_{j_i} + \ldots$$

Since $R_0 \dashrightarrow R_r$, $k+1 \leq r \leq k'$, are unprofitable, $|R_0| \prod_{i=1}^{r-1} P_i(1+P_r) \geq |R_r|$. That is, $\|R_r\| \leq \|R_0\| \prod_{i=1}^{r-1} P_i(1+P_r) \leq \|R_0\| \prod_{i=1}^{r-1} P_{j_i}(1+P_{j_r})$, $k+1 \leq r \leq k'$.

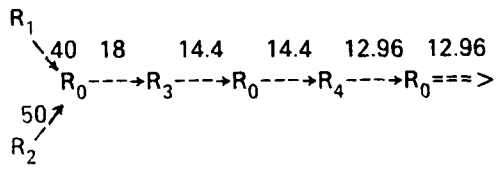
Therefore, $C(\beta) \geq \sum_{i=1}^{k} \|R_{j_i}\| + \sum_{r=k+1}^{k'} \|R_r\| + \|R_0\| \prod_{i=1}^{k'} P_{j_i} + \ldots \geq C(\sigma)$. QED

4.3. A Comparison of Heuristic Algorithms

In example 2, we apply Algorithm OPSTAR to obtain the optimal program

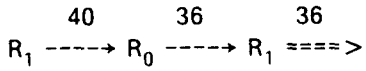


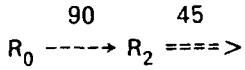
Assume that $|R_0.t| = |R_0|$ and $||R_i.y_j|| = |R_i.y_j|$, where $0 \leq i \leq 4$ and y_j = t, x_1, x_2, x_3 or x_4; The costs for each semi-join execution and data transmissions to the query requesting site (indicated by ====>) are shown as follows:

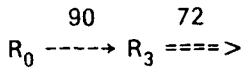


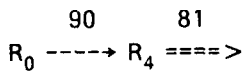
The total cost is 162.72.

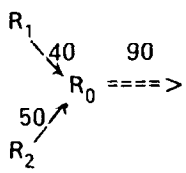
If we apply Algorithm GENERAL(TOTAL) developed by Apers, Hevner and Yao [1] to the same query, then we have the following solution:





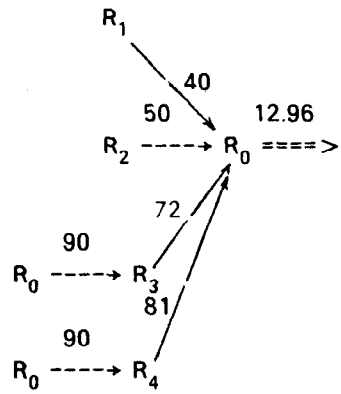






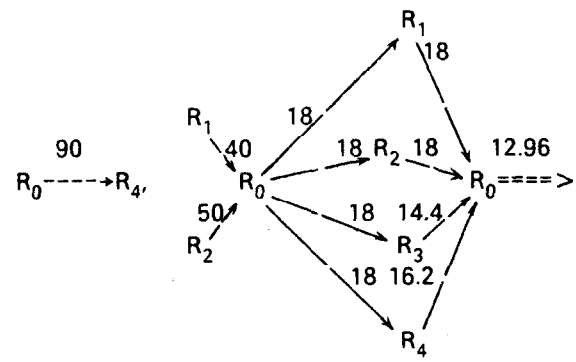
The total cost is 760.

The solution for METHOD-D suggested by Cheung [8] is:



with a total cost of 435.96.

Finally, Algorithm OPT in SDD-1 [2] gave the following solution:



The total cost is 331.56.

For this example, Algorithm GENERAL(TOTAL) generates the worst solution. The major reason is that it transmits all the relations referenced in the query to the query requesting site and it also reduces each relation independently. METHOD-D is an improved algorithm of Algorithm GENERAL(TOTAL). Since it identifies two sets of relations referenced in the query, i.e., the elimination set and the destination set, and transmits only the destination set (in this example, only R_0 is in this set) to the query requesting site. For Algorithm OPT, some redundant semi-joins (e.g., $R_0(0) \rightarrow R_4$) are present in the semi-join program, which cause the total cost to be about twice that of the optimal. Note that if we apply the improvement algorithms proposed in Chen and Li [7], then all of these three solutions can be improved to the optimal.

5. CONCLUSION

In an earlier paper, we have developed a procedure for deriving optimal programs for star queries. In this paper, we have made assumptions about the file sizes and the semi-join selectivities to further simplify this procedure. Epstein and Stonebraker's [13] analysis of distributed database processing strategies shows that good selectivity models are crucial to query processing models. We derive the selectivity of different types of semi-joins in a semi-join program individually based on the database state right before the semi-join under consideration is executed. The selectivity model is therefore consistent in the sense that the effect of prior semi-joins is considered. Since our algorithm can also generate optimal programs which minimize the total data transmission cost for solving simple queries in a star network environment, our approach can be seen as a generalization of Sugihara et. al.'s work [19].

References

1. P.M.G. Apers, A.R. Hevner and S.B. Yao. "Optimization Algorithms for Distributed Queries." *IEEE Trans. on Software Engineering SE-9*, 1 (Jan. 1983), 57-68.

2. Bernstein, P.A., et. al. "Query Processing in a System for Distributed Databases(SDD-1)." *ACM Trans. on Database Systems 6*, 4 (Dec 1981), 602-625.

3. P.A. Bernstein and D.M. Chiu. "Using Semi-Joins to Solve Relational Queries." *JACM 28*, 1 (Jan. 1981), 25-40.

4. P.A. Black and W.S. Luk. A New Hueristic for Generating Semi-Join Programs for Distributed Query Processing. Proc. IEEE COMPSAC, December, 1982, pp. 581-588.

5. A.L.P. Chen and V.O.K. Li. Properties of Optimal Semi-Join Programs for Distributed Query Processing. Proc. IEEE COMPSAC, November, 1983, pp. 476-483.

6. A.L.P. Chen and V.O.K. Li. Deriving Optimal Semi-Join Programs for Distributed Query Processing. Proc. IEEE INFOCOM, April, 1984.

7. A.L.P. Chen and V.O.K. Li. Improving Semi-Join Programs for Distributed Query Processing. To appear in Proc. IEEE COMPSAC, November, 1984.

8. Cheung, T. "A Method for Equijoin Queries in Distributed Relational Databases." *IEEE Trans. on Computer C-31*, 8 (August 1982), 746-751.

9. D.M. Chiu. *Optimal Query Interpretation for Distributed Databases*. Ph.D. Th., Harvard University, December 1979.

10. Chiu, D.M., Bernstein, P.A. and Ho, Y.C. "Optimizing Chain Queries in A Distributed Database System." *SIAM J. COMPUT. 13*, 1 (February 1984), 116-134.

11. E.F. Codd. "A Relational Model of Data for Large Shared Data Banks." *Communications of the ACM 13*, (June 1970), 377-387.

12. E.F. Codd. "Relational Completeness of Data Base Sublanguages." *Data Base Systems, Courant Computer Science Symposia Series 6* (1972), 65-90.

13. Epstein, R. and Stonebraker, M. Analysis of Distributed Data Base Processing Strategies. Proc. Sixth International Conference on Very Large Data Bases, May, 1980, pp. 92-101.

14. G.D. Held, M.R. Stonebraker and E. Wong. INGRES - A Relational Data Base System. Proc. NCC, 1975.

15. A. Hevner. *Query Optimization in Distributed Database Systems*. Ph.D. Th., U. of Minnesota, 1979.

16. A.R. Hevner and S.B. Yao. "Query Processing in Distributed Databases." *IEEE Tran. on Software Eng. SE-5*, 3 (May 1979), 177-187.

17. K.T. Huang. *Query Optimization in Distributed Databases*. Ph.D. Th., M.I.T., December 1982.

18. W.S. Luk and Lydia Luk. Optimizing Semi-Join Programs for Distributed Query Processing. Proc. Second International Conference on Databases, August, 1983.

19. Sugihara, K., et. al. Optimization Algorithms for Processing Simple Queries in Star Networks. Proc. IEEE COMPSAC, November, 1983, pp. 547-554.

20. C.T. Yu, et. al. Promising Approach to Distributed Query Processing. Proc. 7th Berkeley Workshop on Distributed Data Management and Computer Networks, Aug, 1982, pp. 363-390.

21. Yu, C.T. and Chang, C.C. On the Design of A Query Processing Strategy in A Distributed Database Environment. Proc. ACM SIGMOD Int. Conf. on Man. of Data, 1983, pp. 30-39.