A Heuristic Approach to Distributed Query Processing

Jo-Mei Chang

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

In a distributed database environment, finding the optimal strategy which fully reduces all relations referenced by a general tree query, may take exponential time. Furthermore, since reduced relations are to be moved to the final site, the optimal strategy which fully reduces all relations does not give an optimal solution to the problem of minimizing the total transmission cost. For a general query, even with only one join attribute, the problem of finding an optimal strategy to reduce the total data transmission cost has been shown to be NP-hard.

In this paper, a heuristic approach is taken to the distributed query processing problem. Different cost benefit functions are defined based on the nature of the relations involved in the semijoin. The proposed algorithm will produce a sequence of cost beneficial semijoin operations to reduce the total data transmission cost involved in answering a general query. For each join attribute, a two phase reduction process is used. The order in which the semijoins are performed is controlled by the projected size of the join attribute. This algorithm produces optimal sequence of semijoins for simple queries. For general queries, the experimental results, obtained by simulation, indicate a substantial improvement over the SDD-1 query processing algorithm.

1. Introduction

The problem of distributed query processing is to find an efficient or optimal strategy to process queries referencing data at different sites. Answering such a query requires data movement between sites. It usually takes the following steps:[BERNS 81a]:

1. reduce the relations referenced in the query.
2. transmit the reduced relations to one designated site, and then execute the query locally at that site.

The critical optimization problem is to perform the reduction step efficiently. A common assumption in distributed query processing is that the cost of data transmission between nodes is the dominant cost and the cost of the local processing is negligible. The objective of distributed query processing is therefore to process queries with a minimum quantity of inter-site data transfers. To further reduce the size of the database, data from two or more relations must be combined. Semijoins [BERNS 81a] can usually be computed with much less data transmission than a join. It also always reduces the number of tuples of the relation on which it is performed. The problem of distributed query processing thus transforms to generating an efficient or optimal sequence of semijoin operations to reduce the total data transmission cost. Research in this direction can be found in [CHIU 80], [YU 80, 81], [HEVNE 79], [BERNS 81b].

[YU 80] and [YU 80] both use dynamic programming techniques to obtain an optimal sequence of semijoins to fully reduce the output relation for tree queries. However, for general tree queries, finding the optimal strategy which fully reduces relations referenced by the query, may take exponential time [YU 80]. Furthermore, since reduced relations are to be moved to the final site, the optimal strategy which fully reduces all relations does not give an optimal solution to the problem of minimizing total transmission cost in distributed query processing. For a general query, even with only one join attribute, the problem of finding an optimal strategy to reduce the total data transmission cost has been shown to be NP-hard [HEVNE 80], [YU 81].

2. Assumptions

In this paper, relational database systems are considered. Further, it is assumed that the cost of local processing is zero and all possible initial local processing has been performed first. A query Q is of the form of conjunctions of equi-join clauses. All attributes are renamed such that the join attributes in a join clause have the same attribute name. The join clause is of the form R.C = R.C. Each query Q consists of k join attributes: C1,C2,...,Ck, where k can be any number. After the initial local processing, the attributes in each relation are either output attributes or join attributes. Relations referenced in the query are assumed to be located at different sites. Also, when multiple copies of a relation exist, it is assumed that one copy has already been preselected.
The cost measure is defined in terms of the total data transmission cost. The transmission cost of sending X bytes of data from site A to site B is assumed to be $K_0 + K_1 X$, where $K_0$ and $K_1$ are some constants.

Similar to [BERNS 81b], the following assumptions are made for estimating the effect of a semijoin.

1. The distinct values in an attribute of a relation are assumed to be uniformly distributed.

2. If the number of distinct values in one attribute is reduced by a semijoin, the number of distinct values in each of the other attributes in the same relation will also be reduced. The hit-ratio model [YAO 77] is used to estimate the reduction on the other attributes.

It is assumed that the following system parameters are available in the system catalog. For each relation $R_i$, $i=1, ..., M$, $n_i$ is the number of records. $a_i$ is the number of attributes, and $S_i$ is the size of $R_i$ (in bytes).

For each attribute $A_{ij}$, $j=1, ..., a_i$, of relation $R_i$: $p_{ij}$ is the selectivity. $p_{ij} = \frac{u_{ij}}{v_{ij}}$, where $u_{ij}$ is the number of current $A_{ij}$ values in $R_i$, and $v_{ij}$ is the number of possible $A_{ij}$ values. $b_{ij}$ is the projected size of $A_{ij}$. $b_{ij} = w_{ij} * v_{ij}$ (in bytes), where $w_{ij}$ is the size of data item in $A_{ij}$ (in bytes).

3. Cost Benefit Semijoin

3.1 Relevant Set

For a query $Q$, the relevant set of a join attribute $C_k$, $REV(C_k)$, consists of all relations with $C_k$ as the join attribute. For example,

$Q: R_1, C_1 \leftarrow R_2, C_1 \land R_2, C_2 \land R_1, C_2 \leftarrow R_3, C_2$

$REV(C_1) = \{R_1, R_2, R_3\}$

$REV(C_2) = \{R_1, R_3\}$

When $C_k$ is clearly specified in the context, we will use $B_i$ and $P_i$ to represent the projected size and selectivity of $C_k$ for relation $R_i$ in $REV(C_k)$. For each $C_k$, there is a corresponding set of $B_i$ and $P_i$ values associated with each $R_i$ in $REV(C_k)$.

In the above example, if $A_{11} = C_1$, $A_{21} = C_1$, $A_{22} = C_2$, and $A_{31} = C_2$.

In $REV(C_1)$, $P_1 = P_{11}$, $P_2 = P_{21}$.

In $REV(C_2)$, $P_2 = P_{22}$, $P_3 = P_{31}$.

3.2 Effect of a Semijoin

The semijoin [BERNS 81b] of relation $R_i$ with relation $R_j$ on clause $R_i, C = R_j, C$, equals the join of $R_i$ and $R_j$ on that clause projected back onto attributes in $R_j$. The notation $(R_i \rightarrow R_j)$ on $C$ or simply $(R_i \rightarrow R_j)$ will be used interchangeably to denote the semijoin operation. The semijoin $(R_i \rightarrow R_j)$ eliminates the unqualified tuples in $R_i$. $R_i$ is called the input relation and $R_j$ the reduced relation. A sequence of semijoins is called a strategy. For a chain of semijoins defined on $C$, $(R_1 \rightarrow R_2), (R_2 \rightarrow R_3), ..., (R_{j-1} \rightarrow R_j), R_j$ will have the accumulated effect of all the semijoins. The join attribute of $R_j$ is referred to as the accumulated join attribute. The process of performing a sequence of semijoins on $C$ is referred to as accumulating the values of the join attribute $C$. The reduced value of $R_j$, $R_j'$, is referred to as the accumulated projection size of $C$ in $R_j$. The selectivity, $P_j$, of the join attribute of $R_j$ is the probability that an arbitrary value can be found in the join attribute of $R_j$. For a given join attribute $C$, the selectivity of $C$ in $R_j$ is proportional to the projected size of $C$ in $R_j$, $P_j'$ is the accumulated selectivity of the join attribute of $R_j$. That is, $P_j'$ is the selectivity accumulated due to the semijoins. After the semijoin $(R_i \rightarrow R_j)$, the size of $R_i, S_i'$, and the projected size of the join attribute, $B_j'$, is reduced proportionally to the $P_j'$ value. i.e., $B_j' = B_i * P_j'$ and $S_j' = S_j * P_j'$. These values are important for the estimation of the cost and benefit associated with a semijoin. For a sequence of semijoins $(R_i \rightarrow R_j)$ in a given strategy, the estimation of the value of $P_j'$, $B_j'$, and $S_j'$ is described in detail in [CHANG 81].

3.3 Cost Benefit Function

The cost of performing the semijoin $(R_i \rightarrow R_j)$. Cost $(R_i, R_j)$, is the data transmission cost of moving the join attribute from the input relation $R_i$ to the site where $R_j$ is located. Therefore, Cost $(R_i, R_j) = K_0 + K_1 * B_i$, where $B_i$ is the projected size of the join attribute of $R_i$. $K_0$ and $K_1$ are some constants. The benefit due to the semijoin $(R_i \rightarrow R_j)$, BENEFIT $(R_i, R_j)$, is the reduction in the transmission cost due to the size reduction of $R_j$. Therefore, BENEFIT $(R_i, R_j)$ is $K_1 * (S_j - S_j')$, where $S_j$ is the size of relation $R_j$ before the semijoin and $S_j'$ is the reduced size of relation $R_j$ after the semijoin.

A cost benefit function associated with semijoin $(R_i \rightarrow R_j)$, COST-BENEFIT $(R_i, R_j)$, is equal to BENEFIT $(R_i, R_j) - Cost(R_i, R_j)$. A cost beneficial semijoin is a semijoin whose corresponding benefit is greater than its cost, i.e., COST-BENEFIT > 0.

A single attribute relation is a relation which consists of only one join attribute after the local processing, such as select and project, is performed. A multi-attribute relation is a relation which consists of more than one join attribute and/or output attribute after the local processing is performed. The existence of single attribute relations in the query is common. However, it has the following special features to be considered.

1. For a single attribute relation $R_i$, if no semijoin $(R_i \rightarrow R_j)$ is performed, $R_i$ will have to be moved to the final site.

2. For a single attribute relation $R_i$, if semijoin $(R_i \rightarrow R_j)$ is performed, the join operation between $R_i$ and $R_j$ is completed. $R_i$ will no longer be moved to the final site.

Therefore, for a semijoin $(R_i \rightarrow R_j)$, if $R_i$ is a single attribute relation, this would actually save the cost of moving $R_i$ to the final site. The cost of performing such a semijoin is actually zero. Also, if $R_j$ is a single attribute relation and it has already been the input relation of some semijoin, $R_j$ will not be moved to the final site. The benefit of reducing such a relation is actually zero.

Therefore,

1. If $R_j$ is a single attribute relation and it has been used as the input relation of some semijoin, the semijoin with $R_j$ as the reduced relation is non-cost beneficial.

2. Otherwise, if $R_i$ is a single attribute relation, the first semijoin with $R_i$ as the input relation is cost beneficial.

In our approach, we require all the semijoins performed to be cost-beneficial.

4. A Heuristic Algorithm

A heuristic algorithm is proposed to determine the sequence of semijoins used to answer an arbitrary general query $Q$. We divide the reduction process of the query processing into two phases. Phase 1 concentrates on accumulating the values of the join attributes. Phase 2 concentrates on using the accumulated join attribute values to reduce the sizes of the relations. This division allows simple solutions to be found in each phase. To answer a query, this two phase reduction process will be repeated for each join attribute $C_k$.

The reduced relations are used to answer a query, this two phase reduction process will be repeated for each join attribute $C_k$. The reduced relations are
Observation: The cheapest way to accumulate the values of $C_k$ from all relations in $REV(C_k)$ is to continue moving the join attribute which has the smallest (accumulated) projected size to the site which would produce the smallest accumulated projected size [CHANG 81].

Consider the following situation:

$REV(C_k) = \{R_1, R_2, ..., R_k\}$. Relations $R_j$ are renamed so that $B_1 \leq B_2 \leq \cdots \leq B_k$. The cheapest way to accumulate the values of $C_k$ is to perform $(R_1 - R_2), (R_2 - R_3), \cdots, (R_{k-1} - R_k)$. In each of these semijoins, the input relation $R_i$ always has the smallest (accumulated) projected size of $C_k$. The reduced relation $R_j$ is always the relation that will produce the smallest accumulated size of $C_k$ with $R_i$ as the input relation. Each time a semijoin is performed, the relation $R_j$ is selected so that $B_i$ is the smallest among all relations in $REV(C_k)$.

In our approach, all semijoins performed are required to be cost-beneficial. Based on the above observation and the constraint on cost-beneficial semijoins, the following basic rules are used in our algorithm to determine the sequence among all the possible cost beneficial semijoins.

**RULE 1:**
For a given $C_k$ and a $R_i$ in $REV(C_k)$, among all the cost beneficial semijoins ($R_i - R_j$) on $C_k$, always perform ($R_i - R_j$) such that the value of $B_j$, the reduced $B_k$, is the smallest among all $B_j$.

**RULE 2:**
For a given $C_k$, among all the relations in $REV(C_k)$, always try to use the relation $R_i$ which has the smallest projected size of $C_k$ as the input relation. (Only when a cost-beneficial semijoin with $R_i$ as the input relation can be performed, will other relations can be considered as the input relation.)

RULE 1 will determine the reduced relation of a semijoin. RULE 2 will determine the input relation of a semijoin. With RULE 2, between two relations $R_i$ and $R_j$, if $B_i < B_j$, we always first try to perform ($R_i - R_j$). Only when ($R_i - R_j$) is not a cost beneficial semijoin, will we consider ($R_j - R_i$). When all the semijoins are cost-beneficial, RULE 1 and RULE 2 will produce the sequence of semijoins which has the smallest costs among all the possible semijoin sequences which accumulate the values of the $C_k$ from all the relations in $REV(C_k)$.

The algorithm for performing the phase 1 reduction process for a given $C_k$ is described in the following algorithm (ALGORITHM H-1). In ALGORITHM H-1, a list SA is used to record the relations that have been involved in semijoins. A list SI is used to record the relations that are not in SA. Strategy($C_k$) is used to record the resulting semijoins for the phase 1 reduction process of $C_k$. Initially, SA consists of the relation with the smallest $B_i$ on all the relations in $REV(C_k)$. SI consists of the rest of the relations in $REV(C_k)$ in increasing $B_i$ order. ALGORITHM H-1 maintains proper order among the relations in both SA and SI. The first relation in SI always has the smallest $B_i$ value among all the relations in SI. Also the last relation in SA always has the smallest $B_i$ value among all relations in $REV(C_k)$.

**ALGORITHM H-1:**

**Input:** $Q$, $C_k$, $REV(C_k)$.
**Output:** Strategy($C_k$).
**Initialization:** relations $R_i$ in $REV(C_k)$ are renamed so that $B_1 \leq B_2 \leq \cdots \leq B_k$.

**SA = {R_1}, SI = {R_2, ..., R_k}, Strategy(C_k) = empty.**

**STEP 1:** If SI is empty, STOP.
Let $R_m$ be the last element in SA. Scan SI in sequence. Select the first $R_j$ in SI that satisfies $COST\text{-}BENEFIT(R_m, R_j) > 0$. Remove $R_j$ from SI and add to the end of SA. Add $(R_m - R_j)$ to the end of Strategy($C_k$). Repeat STEP 1.

**STEP 2:** If no element in SI satisfies $COST\text{-}BENEFIT(R_m, R_j) > 0$, go to STEP 3.

**STEP 3:** Let $R'_m$ be the first element in SI. Scan SA in sequence. Select the first $R_j$ in SA that satisfies $COST\text{-}BENEFIT(R'_m, R_j) > 0$. Remove $R'_m$ from SA and add to SI right before $R_j$. Add $(R'_m - R_j)$ to Strategy($C_k$) right after the last semijoin reduces $R_j$.
Go to STEP 1.

**STEP 4:** If no element in SA satisfies $COST\text{-}BENEFIT(R'_m, R_j) > 0$ and no single attribute relation in SI, STOP. Otherwise, move the first single attribute relation in SI as the first element in SA, and go to STEP 3.

In ALGORITHM H-1, STEP 1 implements the first alternative of adding elements to SA. STEP 3 implements the second alternative of adding elements to SA. The relative order of the elements in SA and SI are properly maintained by ALGORITHM H-1. Basically, the last element in SA has the smallest $P_i$. Therefore, in STEP 1, if the last element of SA cannot obtain any Cost-Beneficial semijoin with elements in SI, no other element in SA can obtain Cost-Beneficial semijoins with elements in SI. The first element in SI has the smallest projected size of $C_k$. Similarly, in STEP 3, the first element in SI cannot obtain any Cost-Beneficial semijoin with elements in SA. No other element in SI, with the exception of single attribute relations, can obtain Cost-Beneficial semijoins with elements in SA. STEP 4 therefore either stops ALGORITHM H-1 or moves the single attribute relation to the beginning of the SI. (In most of the cases, STEP 3 is rarely used.)

ALGORITHM H-1 will collect the values of the join attribute from each relation in $REV(C_k)$ at most once, i.e. each relation will be the input relation of some semijoin at most once. Due to STEP 3, it is possible taht some relation may be the reduced output.
To further reduce the transmission cost, the project operation is to be performed to eliminate the join attributes that are not to be used for future processing. For a query Q, Ck is Rm-reducible, if (1) there is only one multi-attribute relation Rm in the REV(Ck) and the rest are all single attribute relations, and (2) Ck is not an output attribute. If Ck is Rm-reducible, we can eliminate Ck from Rm at the end of phase 2 and still obtain the correct answer for Q.

The order of which the join attributes are to be processed is determined based on the following observations:

Observation 1:
Since the reduction of one join attribute will indirectly reduce the other attributes in the same relation, one reasonable choice of ordering the join attributes is to first process the join attribute that will most reduce the relations in its relevant set. That is, to first process the join attribute with the smallest value for the product of the selectivity for all the relations in its relevant set.

Observation 2:
For a given Ck, the cost of performing the semijoins on Ck are influenced by the projected sizes of R's in REV(Ck). Another choice of ordering the join attributes is to order the join attributes according to their projected size.

The following heuristic rule is therefore used. For a given Ck, let RANGE_Ck be the number of possible values in the domain of Ck.
Let \( ACC_k = RANGE_Ck \prod_{j \in REV(C_k)} R_j \)
The order of which the join attribute is processed is determined according to its associated \( ACC_k \) value. The join attribute has the smallest \( ACC_k \) value will be processed first.

**Rule 3: Rule of Ordering**
Order join attributes \( C_k \) according to the following priorities:

1. The join attribute has the smallest \( ACC_k \) value has the highest priority.

The algorithm H-1 and H-2 are repeated for each join attribute. The overall strategy for Q, Strategy(Q), is therefore Strategy(Q) = \{ Strategy(C_1), Strategy(C_2),..., Strategy(C_n) \}, where each \( C_i \) is selected dynamically, after \( C_{i-1} \) is selected and Strategy(\( C_{i-1} \)) is produced. Initially, \( C_1 \) is selected as the join attribute with the highest priority.

**4.4 Enchancement Rules and Rule of Shipping**
This subsection discusses the enchancement rules used to further reduce the cost of the semijoins. Also, a specific rule, rule of shipping, is used to specify which relations are to be shipped to the final site in order to form the correct response set for the user queries.

We assume that the reduction of the join attribute will reduce the other attributes in the same relation. Since \( (R_i - R_j) \) will reduce the projected size of other attributes in \( R_i \), besides the join attribute, it would sometimes be beneficial if some semijoins in Strategy(\( C_{i+1} \)) are performed before semijoins in Strategy(\( C_i \)). However, when the semijoins in Strategy(Q) are reordered, the relative order of each of the join attributes in a strategy(\( C_k \)) cannot be changed. That is, if semijoin \( j \) in Strategy(\( C_j \)), then semijoin \( j \) has to be before semijoin \( j \) in Strategy(Q).

Consider the following example:

Strategy(\( C_1 \)) = \{(R_1 - R_2), (R_3 - R_4)\},
Strategy(\( C_2 \)) = \{(R_2 - R_3), (R_3 - R_4)\}.

Since \( (R_2 - R_3) \) reduces \( R_3 \), the cost of \( (R_3 - R_1) \) on \( C_1 \) can be decreased by delaying it until \( (R_2 - R_2) \) on \( C_3 \) is performed. This reordering also does not increase the cost of any other semijoins.

Strategy(Q) = \{(R_1 - R_2), (R_3 - R_4), (R_1 - R_3), (R_2 - R_4)\} has a lower cost than Strategy(Q) = \{(R_1 - R_2), (R_2 - R_1), (R_1 - R_3), (R_1 - R_4)\}.

The permutation rule in [BERNS 81b] permutes the order of semijoins in a given strategy to decrease the cost of semijoins without increasing the cost of any others. This permutation rule will be used to reorder the semijoins in Strategy(Q) to further reduce the total cost. The details of the permutation algorithm, can be found in [BERNS 81b].

**RULE 4: Rule of Transformation**
Using the permutation rule in [BERNS 81b], reorder the semijoins in Strategy(Q).

At the end of the reduction phase, reduced relations are sent to a designated site and the query Q is executed locally at that site. The following rule specifies which relations are to be sent to the final site in order to produce the answer for Q.

**RULE 5: Rule of Shipping**
If all relations referenced by Q are single attribute relations, move the relation which was reduced last to the designated site. Otherwise, move all multi-attribute relations to the designated site.

MOVE(\( R_i, R_j \)) is used to represent the move of the relation \( R_i \) to the site where \( R_j \) is located. For each relation \( R_i \) moved to the final site, MOVE(\( R_i, R_j \)) will be added to the Strategy(Q).

The cost associated with the move operation is \( K_i + K_j + S_i \), where \( S_i \) is the size of the \( R_i \).

In calculating the cost benefit function, the fact that a relation may already be located at the final site was not taken into account.
There are two possibilities for a semijoin \((R_i - R_j)\) on \(C_k\) to be included in the strategy: either in phase 1 or in phase 2 of the reduction process of \(C_k\).

1. If \((R_i \rightarrow R_j)\) occurs during phase 1: After \((R_i \rightarrow R_j)\), there exist semijoin(s) \((R_j \rightarrow R_i)\) on \(C_k\). To remove \((R_i \rightarrow R_j)\) will involve changing all \((R_j \rightarrow R_i)\) to \((R_i \rightarrow R_j)\), where \(k_i \neq k_j\).

2. If \((R_i \rightarrow R_j)\) occurs during phase 2: \((R_i \rightarrow R_j)\) will not be followed by any \((R_j \rightarrow R_i)\) on \(C_k\). Removal of \((R_i \rightarrow R_j)\) will not affect other semijoins.

**RULE 6: Rule of Final Site**

Given a Strategy(Q), for each semijoin \((R_i - R_j)\) on \(C_k\)

1. If \((R_i \rightarrow R_j)\) is followed by semijoin(s) \((R_j \rightarrow R_i)\) on \(C_k\), Strategy(Q) is constructed as follows:
   (a) Remove \((R_i \rightarrow R_j)\).
   (b) Translate each \((R_j \rightarrow R_i)\) into \((R_i \rightarrow R_j)\), if \(R_i \neq R_j\). Otherwise, remove \((R_j \rightarrow R_i)\).
   The cost of the Strategy(Q) is compared to the cost of Strategy(Q) or the Strategy(Q) is replaced with Strategy(Q).

2. If \((R_i \rightarrow R_j)\) is not followed by any semijoin \((R_j \rightarrow R_i)\) on \(C_k\), Strategy(Q) is constructed by removing \((R_i \rightarrow R_j)\) from Strategy(Q). The cost of the Strategy(Q) is compared to the cost of Strategy(Q). If the latter cost is lower, Strategy(Q) is replaced with Strategy(Q).

The overall processing for a general query Q is described in the following algorithm.

**ALGORITHM H-Q:**

Input: Q.
Output: Strategy(Q).
Initialization: Strategy(Q) = empty. JOIN\(_j\) = \{C\(_1\),...,C\(_m\)\}, C\(_i\) are join attributes in Q. k = 1.

**STEP 1:** If JOIN\(_j\) is empty, go to **STEP 4**.

**STEP 2:** Apply Rule of Ordering to select the attribute with the highest priority among attributes in JOIN\(_j\) as C\(_k\).

**STEP 3:** Apply ALGORITHM H-1 and H-2 to produce Strategy(C\(_k\)).

**STEP 4:** Apply Rule of Transformation.

**STEP 5:** Apply Rule of Shipping.

**STEP 6:** Apply Rule of Final Site. STOP.

5. **Examples**

In order to illustrate the algorithm described in Section 4, the following examples are given.

**5.1 Simple Query**

A simple query [HEVNE 79] is defined such that after initial local processing each relation in the query contains only one attribute, namely, the join attribute.

**Example 1:** Let Q be a simple query with relations R\(_i\), i = 1, ..., m. Each R\(_i\) consists of only one attribute C\(_i\).

The relations R\(_i\) are reordered, so that B\(_1\) \leq B\(_2\) \leq \cdots \leq B\(_m\).

- Apply ALGORITHM H-1 and H-2 for C:
  - Since all R\(_i\) are single attribute relations, (R\(_i\) \rightarrow R\(_i+1\)) is always cost beneficial.
  - Strategy(C) = \{(R\(_1\) \rightarrow R\(_2\)), (R\(_2\) \rightarrow R\(_3\)), ..., (R\(_{m-1}\) \rightarrow R\(_m\))\}.

- Since there is only one join attribute, Strategy(Q) = Strategy(C) = \{(R\(_1\) \rightarrow R\(_2\)), (R\(_2\) \rightarrow R\(_3\)), ..., (R\(_{m-1}\) \rightarrow R\(_m\))\}.

- Apply Rule of Shipping and Rule of Final Site:
  - (a) If R\(_m\) = R\(_f\).
    - Since R\(_m\) is the final site, no new rule is added.
    - Strategy(Q) = \{(R\(_1\) \rightarrow R\(_2\)), (R\(_2\) \rightarrow R\(_3\)), ..., (R\(_{m-1}\) \rightarrow R\(_m\))\}.
  - (b) If R\(_m\) \neq R\(_f\).
    - Apply Rule of Shipping. MOVE(R\(_m\),R\(_f\)) is added.
    - Strategy(Q) = \{(R\(_1\) \rightarrow R\(_2\)), (R\(_2\) \rightarrow R\(_3\)), ..., (R\(_{m-1}\) \rightarrow R\(_m\)), MOVE(R\(_m\),R\(_f\))\}.
    - Apply Rule of Final Site, the cost of Strategy(Q) is compared with Strategy(Q), where Strategy(Q) = \{(R\(_1\) \rightarrow R\(_2\)), ..., (R\(_{m-1}\) \rightarrow R\(_f\)), ..., (R\(_{m-1}\) \rightarrow R\(_m\)), MOVE(R\(_m\),R\(_f\))\}.
    - Therefore, Strategy(Q) is equal to
      - (1) if Cost(Strategy(Q)) < Cost(Strategy(Q)), Strategy(Q) = \{(R\(_1\) \rightarrow R\(_2\)), ..., (R\(_{m-1}\) \rightarrow R\(_f\)), ..., (R\(_{m-1}\) \rightarrow R\(_m\)), MOVE(R\(_m\),R\(_f\))\}.
      - (2) otherwise, Strategy(Q) = \{(R\(_1\) \rightarrow R\(_2\)), ..., (R\(_{m-1}\) \rightarrow R\(_f\)), ..., (R\(_{m-1}\) \rightarrow R\(_m\)), MOVE(R\(_m\),R\(_f\))\}.

Since all R\(_i\) are single attribute relations, B\(_i\) = S\(_i\), for all i.

The solution obtained above is actually the optimal solution for a simple query as presented in [HEVNE 79].

**5.2 General Query**

**Example 2:** Given a distributed database with four relations, E: EMPLOYEE (E\# , ENAME, SEX), C: COURSE (C\# , CNAME, LEVEL), SC: STUDENT-COURSE (E\# , C\#), TC: TEACHER-COURSE (E\# , C\#, ROOM) [HEVNE 79].

Assume that the site containing the TEACHER-COURSE relation is the result site. Consider the following query Q:

- for all male employees who are teaching advanced courses in Room 103 and are students in at least one course, list the employees’ names and the courses they are teaching.

The first step is to do local processing. The local restrictions on E.SEX, C.LEVEL, and TC.ROOM are performed and the required joining attribute and output attributes E.ENAME and C.CNAME are projected.

The qualification of Q is (E.SEX = 'M') \land (SC.C\# = TC.E\#) \land (TC.C\# = C.C\#).

The target list of Q is E.ENAME and C.CNAME.

The parameters associated with the relations are given as follows.

<table>
<thead>
<tr>
<th>Relation</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>600</td>
<td>200</td>
<td>1/5</td>
<td>200</td>
</tr>
<tr>
<td>C</td>
<td>600</td>
<td>600</td>
<td>3/5</td>
<td>1200</td>
</tr>
<tr>
<td>SC</td>
<td>2000</td>
<td>200</td>
<td>1/5</td>
<td>100</td>
</tr>
<tr>
<td>TC</td>
<td>100</td>
<td>1/4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let $K_0 = 10, K_1 = 1$.

- **Apply Rule of Ordering:**
  \[
  \text{REV}(E#) = [TC, SC, E], \text{REV}(C#) = [TC, C].
  \]
  $\text{ACC}(E#) < \text{ACC}(C#)$. $E#$ will be processed first.

- **Phase 1 Reduction for $E#$:** $B_E < B_{TC} < B_{SC}$.
  (a) first try ($E \rightarrow TC$):
  \[
  \text{COST-BENEFIT}(E#, TC) = (1-1/5)*600 - (200+10) > 0
  \]
  Strategy($E#$) = $[(E \rightarrow TC)]$.
  (b) try ($TC \rightarrow SC$):
  \[
  \text{COST-BENEFIT}(TC, SC) = (1-1/25)*60 - (40+10) > 0
  \]
  Strategy($E#$) = $[(TC \rightarrow SC)]$.

- **Phase 2 Reduction for $E#$:**
  \[
  \text{COST-BENEFIT}(SC, E) > 0, \text{COST-BENEFIT}(SC, TC) > 0.
  \]
  Strategy($E#$) = $[(E \rightarrow TC), (TC \rightarrow SC), (SC \rightarrow E), (SC \rightarrow TC)]$.

- **Phase 1 Reduction for $C#$:**
  $B'_{TC} = 40$ (Using hit ratio model). $B'_{TC} < B_C$.
  first try ($TC \rightarrow C$):
  \[
  \text{COST-BENEFIT}(TC, C) > 0.
  \]
  Strategy($C#$) = $[(TC \rightarrow C)]$.

- **Phase 2 Reduction process for $C#$:**
  \[
  \text{COST-BENEFIT}(C, TC) > 0.
  \]
  Strategy($C#$) = $[(TC \rightarrow C)]$.

- **Strategy($Q$) = $[(E \rightarrow TC), (TC \rightarrow SC), (SC \rightarrow E), (SC \rightarrow TC), (TC \rightarrow C), (C \rightarrow TC)]$.**

- **Apply Rule of Transformation:**
  Rule of ordering does not change the order of the semijoins in Strategy($Q$).

- **Apply Rule of Shipping:**
  Strategy($Q$) = $[(E \rightarrow TC), (TC \rightarrow SC), (SC \rightarrow E), (SC \rightarrow TC), (TC \rightarrow C), (C \rightarrow TC), MOVE(E, TC), MOVE(C, TC)]$.

- **Apply Rule of Final Site:**
  TC is at the final site, rule of final site will remove $SC \rightarrow TC, C \rightarrow TC$ from the strategy. Therefore,
  Strategy($Q$) = $[(E \rightarrow TC), (TC \rightarrow SC), (SC \rightarrow E), (TC \rightarrow C), MOVE(E, TC), MOVE(C, TC)]$.

This presents the correct solution to the problem suggested in [HEVNE 79]. The solution suggested by [HEVNE 79] is $[(E \rightarrow TC), (TC \rightarrow SC), (SC \rightarrow E), (C \rightarrow TC), MOVE(E, TC), MOVE(C, TC)]$. They failed to recognize that the size of relation TC had been considerably reduced by the reduction process of $E#$. The semijoin $(C \rightarrow TC)$ as suggested in their solution is not a cost-beneficial move and therefore their solution is incorrect according to their own problem formulation.

**Example 3:** Consider the same query as in Example 2. The qualification of $Q$ is
\[
(E.E# = SC.E#) \land (SC.E# = TC.E#) \land (TC.C# = C.C#).
\]
However, the target list consists only $E.E#$. Also, $TC$ is assumed to be located at the final site [YU 80]. The parameters associated with the relations are given as follows.

<table>
<thead>
<tr>
<th>$R_i$</th>
<th>$S_i$</th>
<th>$B_i(E#)$</th>
<th>$P_i(E#)$</th>
<th>$B_i(C#)$</th>
<th>$P_i(C#)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC</td>
<td>600</td>
<td>300</td>
<td>3/4</td>
<td>300</td>
<td>1/2</td>
</tr>
<tr>
<td>SC</td>
<td>240</td>
<td>240</td>
<td>3/5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>300</td>
<td>300</td>
<td></td>
<td>300</td>
<td>1/2</td>
</tr>
</tbody>
</table>

Let $K_0 = 10, K_1 = 1$.

- **Apply Rule of Ordering:**
  \[
  \text{REV}(E#) = [TC, SC, E], \text{REV}(C#) = [TC, C].
  \]
  $\text{ACC}(E#) < \text{ACC}(C#)$. $E#$ will be processed first.

- **Apply ALGORITHM H-1 and H-2 for $E#$:**
  \[
  B'_{TC} = 180 \text{ (according to hit ratio model).}
  \]
  Strategy($E#$) = $[(E \rightarrow SC), (SC \rightarrow TC)]$.

- **Apply ALGORITHM H-1 and H-2 for $C#$:**
  \[
  B'_{TC} < B_C.
  \]
  Strategy($C#$) = $[(TC \rightarrow C), (C \rightarrow TC)]$.

**Strategy($Q$) = $[(E \rightarrow SC), (SC \rightarrow TC), (TC \rightarrow C), (C \rightarrow TC)]$.**

- **Apply Rule of Transformation. Rule of shipping and Rule of Final Site:**
  Since TC is the final site, Strategy($Q$) is the same.
  \[
  \text{Cost (Strategy($Q$))} = \text{Cost(E, SC)} + \text{Cost(SC, TC)} + \text{Cost(TC, C)} + \text{Cost(C, TC)} = (10 + 200) + (10 + 120) + (10 + 180) + (10 + 90) = 630.
  \]

Using the algorithm in SDD-1, their strategy is to perform the semijoin which maximizes the immediate gain. According to the cost benefit definition in [BERNS 81b], the only cost beneficial semijoin is $(E \rightarrow TC)$. Therefore, their solution would be:
\[
\text{Strategy(SDD-1)} = [(E \rightarrow TC), MOVE(SC, TC), MOVE(C, TC)].
\]
This is also the solution that the algorithm in [HEVNE 79] would suggest.
\[
\text{Cost(Strategy(SDD-1))} = \text{Cost(E, TC)} + \text{Size(SC)} + \text{Size(C)} = (10 + 200) + (10 + 240) + (10 + 300) = 870.
\]
This is much higher than our strategy.

6. **Experimental Results**

Simulation programs were written to compare the performance of our proposed heuristic algorithm and SDD-1 query processing algorithm. For a given query, separate sequences of semijoins are generated according to the SDD-1 algorithm and our algorithm. The costs of performing these sequences of semijoins are then calculated. Both the SDD-1 strategy and our strategy have been applied under the following conditions: (1) the rule of transformation and rule of final site have not been used. (2) after the reduction phase, the site with the largest relation size is dynamically chosen to be the final site.

The performance improvement of our algorithm over SDD-1 algorithm is calculated as follows:
\[
\text{improvement} = \frac{(\text{Cost(Strategy(SDD-1))} - \text{Cost(Strategy(Q))})}{\text{Cost(Strategy(SDD-1))}}.
\]
For each query, the corresponding system parameters (the number of tuples, the number of attributes in each relation, and the selectivity associated with each join attribute) are randomly generated. The average improvement for a query $Q$ is calculated as the average improvement of $Q$ tested over 500...
different combinations of relation sizes, possible join attribute values and selectivities.

We have empirically tested the common query patterns, i.e. queries consisting of one, two and three join attributes. These query patterns included cyclic queries as well as tree queries. The empirical results indicate up to 50% performance improvement of our algorithm over SDD-1. The improvement over SDD-1 increases in general with the increase of (1) the number of relations referenced in the query, (2) the percentage of single attribute relations referenced in the query.

Let M be the total number of relations referenced in the query and S be the number of single attribute relation referenced in the query. For each query pattern, experiments have been performed for different combinations of M and S values. The characteristics of the corresponding system parameters used in each experiment are described in Table 1. When a wider range of relation sizes and join attribute values is used in the experiments, similar results have also been obtained.

Figure 1 gives the average improvement of our algorithm over SDD-1 algorithm when the query consists of one join attribute.

\[ Q_1: (R_0.A = R_1.A) \land (R_0.A = R_1.A) \land (R_0.A = R_{M-1}.A) \]

Experiments have been performed for M = 3, 4, 5 and for S = 0,..., M.

Queries consisting of two join attributes are generated from \( Q_2 \):

\[ Q_2: (R_0.A = R_1.A) \land (R_0.B = R_2.B) \land \text{equi-join-clause}_1 \land \text{equi-join-clause}_{1+} \land \ldots \]

where \( \text{equi-join-clause}_i \) is in the form of \((R_0.A = R_1.A)\) or \((R_0.B = R_1.B)\).

Figure 2 shows the improvement in cases where the total number of relations varies from 4 to 6 and the number of single attribute relations varies from 0 to M-1.

Queries consisting of three join attributes are generated from the following two query patterns.

\[ Q_3: (R_0.A = R_1.A) \land (R_0.B = R_2.B) \land (R_0.C = R_3.C) \land \text{equi-join-clause}_1 \land \text{equi-join-clause}_{1+} \land \ldots \]

where \( \text{equi-join-clause}_i \) is in the form of \((R_0.A = R_1.A)\) or \((R_0.B = R_1.B)\) or \((R_0.C = R_1.C)\).

\[ Q_{32}: (R_0.A = R_1.A) \land (R_0.B = R_2.B) \land (R_0.C = R_3.C) \land \text{equi-join-clause}_1 \land \text{equi-join-clause}_{1+} \land \ldots \]

where \( \text{equi-join-clause}_i \) is in the form of \((R_0.A = R_1.A)\) or \((R_0.B = R_1.B)\) or \((R_0.C = R_1.C)\).

Figure 3 shows the average improvement over SDD-1 when \( Q_2 \) is used. In Figure 3, M varies from 4 to 6 and S varies from 0 to M-4. Figure 4 shows the average improvement over SDD-1 when \( Q_{32} \) is used. In Figure 4, M varies from 4 to 6 and S varies from 0 to M-3. \( Q_{32} \) in fact generates cyclic queries.

7. Conclusions

To summarize, we have proposed a distributed query processing algorithm, which produces a sequence of semijoins for general queries. In our algorithm, the CUST-BENEFIT definition has been modified to reflect the special feature of single attribute relations. Also, a two phase reduction process was used. Phase 1 concentrates on accumulating the values of the join attributes. Phase 2 concentrates on using the accumulated join attribute values to reduce the sizes of the relations. This reduction process allows simple solutions to be found in each phase.

This algorithm produces optimal solution for simple queries. For general queries, the empirical results indicate up to 50% performance improvement over SDD-1.

REFERENCES


Table 1: Descriptions of system parameters.

M: 3,4,5 --Figure 1
4,5,6 ---Figure 2,3,4

p_i: unif(0,1) ---Figure 1,2,3,4

q_{ij}: unif(500,10000) --Figure 1, 2, 3, 4
unif(1000,10000) -- similar results were obtained when this value range was used

n_i: unif(100,10000) -- Figure 1, 2, 3, 4
unif(100,10000) -- similar results were obtained when this value range was used

where unif(n,m) indicates the value is uniformly distributed between n and m.
Improvement over SDD-1

Figure 1 Improvement over SDD-1 for queries with one join attribute (21)

Improvement over SDD-1

Figure 2 Improvement over SDD-1 for queries with two join attributes (22)

Improvement over SDD-1

Figure 3 Improvement over SDD-1 for tree queries with three join attributes (23)

Improvement over SDD-1

Figure 4 Improvement over SDD-1 for cyclic queries with three join attributes (24)

Proceedings of the Eighth International Conference on Very Large Data Bases

Mexico City, September, 1982