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 81b]:

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].

[CHIU 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]. (General queries include both tree queries and cyclic queries). [HEVNE 79] presents an algorithm that constructs an optimal sequence of semijoins for solving a given simple query. (A simple query references only single domain relations). However, their algorithm for a general query, uses an improved exhaustive search and does not consider the fact that the reduction of one join attribute will indirectly reduce the other attributes in the same relation. [BERNS 81b] proposes a heuristic algorithm for answering general queries in SDD-1 [ROTHN 80]. Although the heuristics are simple, the sequence of semijoins generated is in general suboptimal. In this paper, we propose a simple yet efficient heuristic algorithm to generate a sequence of semijoin operations to reduce the total transmission cost of answering a general query. In section 2, the assumptions are made. In section 3, the costs and benefits associated with a semijoin are examined. In section 4, the heuristic algorithm is described. In section 5, examples are given to illustrate the algorithm. In section 6, experimental results are given. Some concluding remarks are given in Section 7.

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 $R_i.C = R_j.C$. Each query Q consists of k join attributes: $C_1, C_2, \ldots, C_k$, 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*. $p_{ij}=u_{ij}/v_{ij}$, 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}. $b_{ij}=u_{ij}*w_{ij}$ (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: $R_1.C_1 = R_2.C_1 \wedge R_2.C_1 = R_3.C_1 \wedge R_1.C_2 = R_3.C_2$.
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_j. 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 B_j, B'_j, is referred to as the accumulated project 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_i is proportional to the projected size of C in R_i. P'_j is the accumulated selectivity of the join attribute of R_j. That is , P' is the selectivity accumulated due to the semijoins. After the semijoin ($R_i \rightarrow R_j$), the size of R_j, S'_j, and the projected size of the join attribute, B'_j, is reduced proportionally to the P'_j value. i.e., $B'_j = B_j * P'_j$, and $S'_j = S_j * P'_j$. These values are important for the estimation of *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 ($R_1 \rightarrow R_2$), 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, with the following exceptions.

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

then sent to the final site, and Q is executed at that site.

4.1 PHASE 1 Reduction Process

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)=$\{R_1, R_1, ..., R_k\}$.
relations R_i 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 $(R_1 \rightarrow R_2)$, $(R_2 \rightarrow R_3)$, $\cdots$ $(R_{k-1} \rightarrow R_k)$. 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 with the smallest (accumulated) projected size of C_k changes dynamically. After each semijoin, since the relation R_j has the accumulated effect of the semijoins performed, it has the current smallest accumulated projected size of C_k. R_j will be used as the input relation for the next semijoin.

In our approach, all semijoins performed are required to be cost-beneficial. Based on the above observation and the contraint 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 $(R_i \rightarrow R_j)$ on C_k, always perform $(R_i \rightarrow R_k)$ such that the value of B'_k, 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 no 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 $(R_i \rightarrow R_j)$. Only when $(R_i \rightarrow R_j)$ is not a cost beneficial semijoin,will we consider $(R_j \rightarrow R_i)$. 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 among 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). There are two alternatives that a relation can be added to SA. The first alternative is to semijoin the relation in SA with the relation in SI. Since P_i is proportional to B_i, performing the semijoin between the last relation of SA and the first relation of SI which satisfies the cost-beneficial requirement will always produce the smallest accumulated projected size of C_k. RULE 1 is thus obeyed. Since the last relation in SA always the smallest B_i value, RULE 2 is also obeyed. This process will continue until all the possible semijoins are exhausted. Because of the constraint on cost beneficial semijoins, SA may not yet have all the relations in REV(C_k). The second alternative of adding relations to SA is to semijoin the relation in SI with the relation in SA. Since the relations in SI will always have larger projected size of C_k than relations in SA, according to RULE 2, this alternative will only be considered when the first alternative is exhausted.

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_{in} be the last element in SA. Scan SI in sequence. Select the first R_j in SI that satisfies COST-BENEFIT(R_{in},R_j) >0.
Remove R_j from SI and add to the end of SA. Add $(R_{in} \rightarrow R_j)$ to the end of Strategy(C_k).
Repeat STEP 1.

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

STEP 3: Let R'_{in} be the first element in SI. Scan SA in sequence. Select the first R_j in SA that satisfies COST-BENEFIT(R'_{in},R_j) > 0 .
Remove R'_{in} from SI and add to SA right before R_j.
Add $(R'_{in} \rightarrow R_j)$ 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-BENEFIT(R'_{in},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 SI, 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, if 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

relation of some semijoin more than once. In an improved version of ALGORITHM H-1, STEP 3 is modified such that all relations in REV(C_k) will be the input/reduced relation of some semijoin at most once. For the detail of the modified algorithm, please refer to [CHANG 81].

4.2 PHASE 2 Reduction Process

In phase 2 of the reduction process, the accumulated join attribute will be used to further reduce a multi-attribute relation when the corresponding cost benefit function is greater than 0. For a single attribute relation R_s, since the cost of performing the first semijoin with R_s as the input relation is 0, its join attribute value will always have been collected during phase 1. It is therefore not necessary to further reduce any single attribute relation in Phase 2.

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, C_k is R_m-*reducible*, if (1) there is only one multi-attribute relation R_m in the REV(C_k) and the rest are all single attribute relations, and (2) C_k is not an output attribute. If C_k is R_m-reducible, we can eliminate C_k from R_m at the end of phase 2 and still obtain the correct answer for Q.

The algorithm for performing the phase 2 reduction process is described in ALGORITHM H-2.

ALGORITHM H-2:

Input: SA, Strategy(C_k).
Output: Strategy(C_k).

STEP 1: Let R_{in} be the last element in SA. Perform ($R_{in} \rightarrow R_j$) for every multi-attribute relation R_j satisfying COST-BENEFIT(R_{in},R_j) >0.
Add ($R_{in} \rightarrow R_j$) to the end of Strategy(C_k).

STEP 2: Eliminate C_k from R_m, if C_k is R_m-reducible.

4.3 Rule of Ordering

To answer a query Q, the two phase reduction process is repeated for every join attribute C_k to obtain Strategy(C_k). 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 C_k, the cost of performing the semijoins on C_k are influenced by the projected sizes of R_is in REV(C_k). Another choice of ordering the join attributes is to order the join attributes according to its projected size.

The following heuristic rule is therefore used. For a given C_k, let RANGE_Ck be the number of possible values in the domain of C_k.
Let ACC_k = RANGE_Ck * $\prod_{j \in REV(Ck)} P_j$.
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_i according to the following priorities:
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_k)}, 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 reponse set for the user queries.

We assume that the reduction of the join attribute will reduce the other attributes in the same relation. Since ($R_i \rightarrow R_j$) will reduce the projected size of other attributes in R_j 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 semijoins in a strategy(C_k) cannot be changed. That is, if semijoin i is before semijoin j in Strategy(C_k), then semijoin i has to be before semijoin j in Strategy(Q).

Consider the following example:

Strategy(C_1) ={($R_1 \rightarrow R_2$), ($R_2 \rightarrow R_1$)},
Strategy(C_2) ={($R_2 \rightarrow R_3$), ($R_3 \rightarrow R_2$)}.

Since ($R_3 \rightarrow R_2$) reduces R_2, the cost of ($R_2 \rightarrow R_1$) on C_1 can be decrease by delaying it until ($R_3 \rightarrow R_2$) on C_2 is performed. This reordering also does not increase any other semijoins. Strategy(Q_1)={($R_1 \rightarrow R_2$), ($R_2 \rightarrow R_3$), ($R_3 \rightarrow R_2$), ($R_2 \rightarrow R_1$)} has a lower cost than Strategy(Q) ={($R_1 \rightarrow R_2$), ($R_2 \rightarrow R_1$), ($R_3 \rightarrow R_1$), ($R_1 \rightarrow R_3$)}.

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_f) will be added to the Strategy(Q). The cost associated with the move operation is $K_0 + K_1 * S_i$, 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. If a relation R_f is located at the final site, there is no need to move it and moreover a reduction in its size does not directly contribute to the total cost reduction. To determine whether to include the semijoin $(R_i \rightarrow R_f)$ in the strategy, where R_i is any relation and R_f is a relation located at the final site, we compare the cost of the two strategies with and without the semijoin.

There are two possibilities for a semijoin $(R_i \rightarrow R_f)$ 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 $(R_i \rightarrow R_f)$ occurs during phase 1: After $(R_i \rightarrow R_f)$, there exist semijoin(s) $(R_f \rightarrow R_j)$ on C_k. To remove $(R_i \rightarrow R_f)$ will involve changing all $(R_f \rightarrow R_j)$ to $(R_i \rightarrow R_j)$, where $R_i \neq R_j$.

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

RULE 6: Rule of Final Site
Given a Strategy(Q), for each semijoin $(R_i \rightarrow R_f)$ on C_k

(1) If $(R_i \rightarrow R_f)$ is followed by semijoin(s) $(R_f \rightarrow R_j)$ on C_k, Strategy(Q_1) is constructed as follows:

(a) Remove $(R_i \rightarrow R_f)$.

(b) Translate each $(R_f \rightarrow R_j)$ into $(R_i \rightarrow R_j)$,if $R_i \neq R_j$. Otherwise, remove $(R_f \rightarrow R_j)$.

The cost of the Strategy(Q) is compared to the cost of the Strategy(Q_1). If the latter cost is lower, Strategy(Q) is replaced with Strategy(Q_1).

(2) If $(R_i \rightarrow R_f)$ is not followed by any semijoin $(R_f \rightarrow R_j)$ on C_k, Strategy(Q_1) is constructed by removing $(R_i \rightarrow R_f)$ from Strategy(Q). The cost of the Strategy(Q) is compared to the cost of the Strategy(Q_1). If the latter cost is lower, the semijoin is removed from Strategy(Q). Strategy(Q) is replaced with Strategy(Q_1).

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).
Strategy(Q) = {Strategy(Q), Strategy(C_k)}.
Remove C_k from $JOIN_j$, k = k + 1.
go to STEP 1.

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.

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_1) is compared with Strategy(Q), where
  Strategy(Q_1) = $\{(R_1 \rightarrow R_2), ..., (R_{f-1} \rightarrow R_{f+1}), ..., (R_{m-1} \rightarrow R_m)$,MOVE($R_m$,$R_f$)$\}$.
  Therefore, Strategy(Q) is equal to
  (1) if Cost(Strategy(Q)) < Cost(Strategy(Q_1)),
  Strategy(Q)=$\{(R_1 \rightarrow R_2), ..., (R_{f-1} \rightarrow R_f), (R_f \rightarrow R_{f+1}), ..., (R_{m-1} \rightarrow R_m)$,MOVE($R_m$,$R_f$)$\}$.
  (2) otherwise,
  Strategy(Q)=$\{(R_1 \rightarrow R_2), ..., (R_{f-1} \rightarrow R_{f+1}), ..., (R_{m-1} \rightarrow R_m)$,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 at 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.E#=SC.E#) $\wedge$ (SC.E#=TC.E#) $\wedge$ (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.

R_i	S_i	B_i(E#)	P_i(E#)	B_i(C#)	P_i(C#)
TC	600	200	1/5	200	1/2
SC	600	600	3/5		
C	1200			100	1/4
E	2000	200	1/5		

Let $K_0 = 10, K_1 = 1$.

- Apply Rule of Ordering:
  REV(E#) = {TC, SC, E}, REV(C#) = {TC, C}.
  ACC(E#) < ACC(C#). E# will be processed first.
- Phase 1 reduction for E#: $B_E < B_{TC} < B_{SC}$.
  (a) first try (E → TC):
  COST-BENEFIT(E, TC) = (1-1/5)*600 −(200+10) >0
  Strategy(E#)={(E → TC)}.
  (b) try (TC → SC):
  COST-BENEFIT(TC, SC) = (1-1/25)*60 − (40+10) >0
  Strategy(E#)={(E → TC), (TC → SC)}.
- Phase 2 Reduction for E#:
  COST-BENEFIT(SC, E) > 0, COST-BENEFIT(SC, TC) > 0.
  Strategy(E#)={(E → TC), (TC → SC), (SC → E), (SC → TC)}.
- Phase 1 reduction for C#:
  B'_{TC}=40.(Using hit ratio model). $B'_{TC} < B_C$.
  first try (TC → C):
  COST-BENEFIT(TC, C) > 0.
  Strategy(C#) = {(TC → C)}.
- Phase 2 reduction process for C#:
  COST-BENEFIT(C, TC) > 0.
  Strategy(C#)= {(TC → C), (C → TC)}.
- Strategy(Q)= {(E → TC), (TC → SC), (SC → E), (SC → TC), (TC → C), (C → TC)}.
  Applying Rule of Transformation does not change the order of the semijoins in Strategy(Q).
- Apply Rule of Shipping:
  Strategy(Q)= {(E → TC), (TC → SC), (SC → E), (SC → TC), (TC → C), (C → 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 → TC), (C → TC) from the strategy. Therefore,
  Strategy(Q)= {(E → TC), (TC → SC), (SC → E), (TC → 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 → TC), (TC → SC), (SC → E), (C → TC), (TC → C), 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 → 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#) ∧ (SC.E# =TC.E#) ∧ (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.

R_i	S_i	B_i(E#)	P_i(E#)	B_i(C#)	P_i(C#)
TC	600	300	3/4	300	1/2
SC	240	240	3/5		
C	300			300	1/2
E	200	200	1/2		

Let $K_0 = 10, K_1 = 1$.

- Apply Rule of Ordering:
  REV(E#) = {TC, SC, E}, REV(C#) = {TC, C}.
  ACC(E#) < ACC(C#), E# will be processed first.
- Apply ALGORITHM H-1 and H-2 for E#:
  $B_E < B_{SC} < B_{TC}$.
  Strategy(E#)={(E → SC), (SC → TC)}.
- Apply ALGORITHM H-1 and H-2 for C#:
  B'_{TC} = 180 (according to hit ratio model).
  $B'_{TC} < B_C$.
  Strategy(C#)={(TC → C), (C → TC)}.

  Strategy(Q)={(E → SC), (SC → TC), (TC → C), (C → TC) }.
- Apply Rule of Transformation , Rule of shipping and Rule of Final Site:
  Since TC is the final site, Strategy(Q) is the same.
  Cost (Strategy(Q)) = Cost(E,SC) + Cost(SC,TC) + Cost(TC,C) + 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 → TC). Therefore, their solution would be:
Strategy(SDD-1)= {(E → TC), MOVE(SC, TC), MOVE(C, TC)}.
This is also the solution that the algorithm in [HEVNE 79] would suggest.
Cost(Strategy(SDD-1))=Cost(E, TC) + Size(SC) + 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:

improvement = (Cost(Strategy(SDD-1)) − Cost(Strategy(Q)))/ 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: $(R_0.A = R_1.A) \wedge (R_0.A = R_2.A) \wedge (R_0.A = R_{M-1}.A)$

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: $(R_0.A = R_1.A) \wedge (R_0.B = R_2.B) \wedge equi\text{-}join\text{-}clause_i \wedge equi\text{-}join\text{-}clause_{i+1} \wedge \ldots$
where $equi\text{-}join\text{-}clause_i$ is in the form of $(R_0.A = R_i.A)$ or $(R_0.B = R_i.B)$

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_{31}: $(R_0.A = R_1.A) \wedge (R_0.B = R_2.B) \wedge (R_0.C = R_3.C) \wedge equi\text{-}join\text{-}clause_i \wedge equi\text{-}join\text{-}clause_{i+1} \wedge \ldots$

where $equi\text{-}join\text{-}clause_i$ is in the form of $(R_0.A = R_i.A)$ or $(R_0.B = R_i.B)$ or $(R_0.C = R_i.C)$.

Q_{32}: $(R_0.A = R_1.A) \wedge (R_0.B = R_2.B) \wedge (R_1.C = R_2.C) \wedge equi\text{-}join\text{-}clause_i \wedge equi\text{-}join\text{-}clause_{i+1} \wedge \ldots$

where $equi\text{-}join\text{-}clause_i$ is in the form of $(R_0.A = R_i.A)$ or $(R_2.B = R_i.B)$ or $(R_1.C = R_i.C)$.

Figure 3 shows the average improvement over SDD-1 when Q_{31} 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 COST-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

[BERNS 81a] P. A. Bernstein and D. M. Chiu, "Using Semi-joins to Solve Relational Queries", JACM 1981.

[BERNS 81b] P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve and J. B. Rothnie, "Query Processing in A System For Distributed Databases (SDD-1)", ACM TODS, Dec 1981.

[CHIU 80] D. M. Chiu and Y. C. Ho, "A Methodology For Interpreting Tree Queries Into Optimal Semi-Join Expressions", Proceeding ACM SIGMOD 1980.

[CHANG 81] J. M. Chang "Distributed Query Processing: A Heuristic Approach", submitted for publication.

[HEVNE 79] A. R. Hevner and S. B. Yao, "Query Processing in Distributed Database Systems", IEEE Transactions on Software Engineering, May 1979. p.177-p187.

[HEVNE 80] A. R. Hevner, "The optimization of Querying Processing on Distributed Database Systems", Ph.D. Thesis, Dept. of Computer Science, Purdue University, 1980.

[ROTHN 80] J. B. Rothnie, Jr., P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. W. Shipman, and E. Wong, "Introduction to a System for Distributed Databases (SDD-1)", ACM TODS, March 1980.

[YAO 77] S. B. Yao, "Approximating Block Accesses in Database organizations", CACM April 1977.

[YU 80] C. T. Yu, K. Lam and M. Z. Ozsoyoglu, "Distributed Query Optimization for Tree Queries", Technical Report, Dept. of Information Engineering, U.I.C.C. July, 1980.

[YU 81] C. T. Yu, K. Lam, C. C. Chang and S. K. Chang, "Promising Approach to Distributed Query Processing", Proceeding of the 6th Berkeley Workshop on Distributed Data Management& Computer Networks, Feb 1982.

Table 1: Descriptions of system parameters.

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

p_{ij}: unif(0,1) ---Figure 1,2,3,4

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

n_i: unif(100,1000) -- 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.

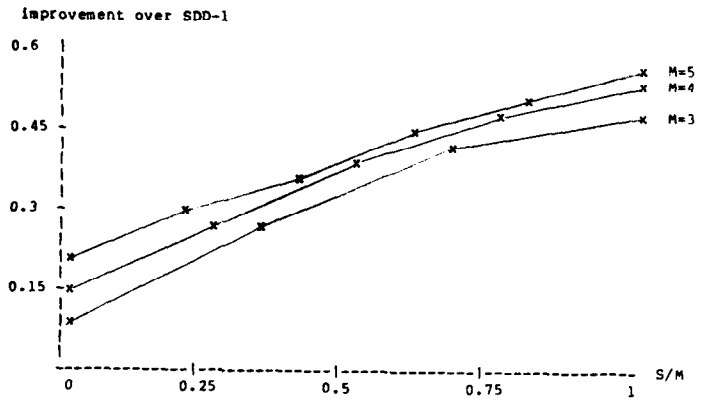

Figure 1 Improvement over SDD-1 for queries with one join attribute (Q1)

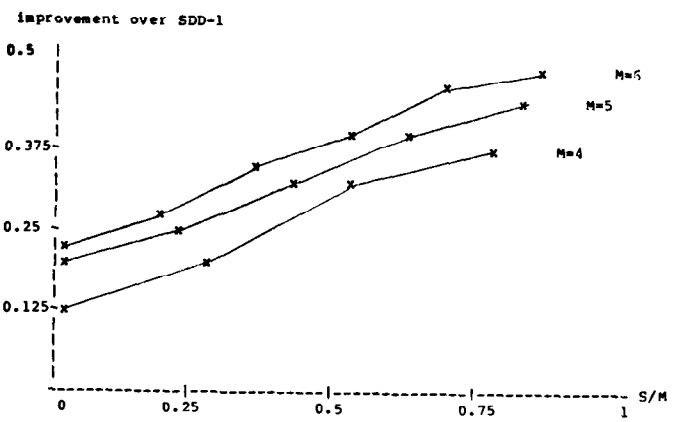

Figure 2 Improvement over SDD-1 for queries with two join attributes (Q2)

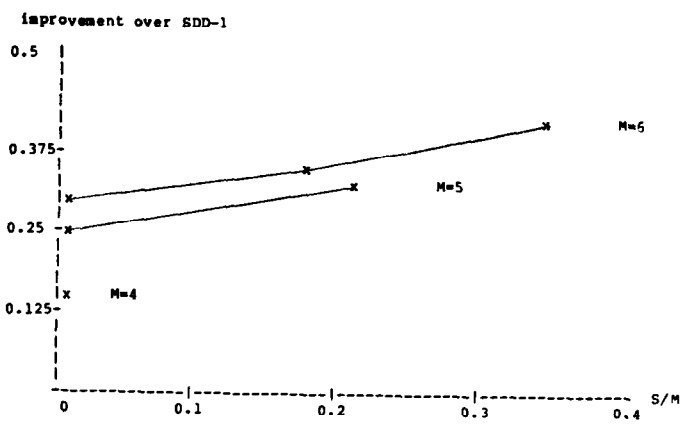

Figure 3 Improvement over SDD-1 for tree queries with three join attributes (Q31)

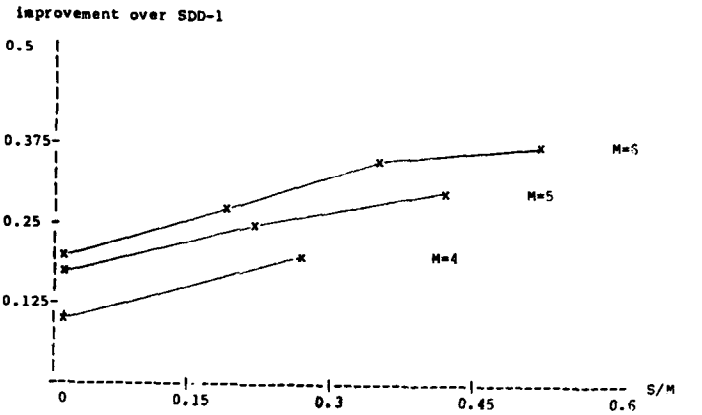

Figure 4 Improvement over SDD-1 for cyclic queries with three join attributes (Q32)