Optimizing Join Queries in Distributed Databases

SAKTI PRAMANIK AND DAVID VINEYARD

Abstract—**A reduced cover set of the set of full reducer semijoin programs for an acyclic query graph for a distributed database system is given. An algorithm based on this reduced cover set is then presented which determines the minimum cost full reducer program. We show that the computational complexity of finding the optimal full reducer for a single relation is of the same order as that of finding the optimal full reducer for all the relations. The optimization algorithm is able to handle query graphs where more than one attribute is common between the relations. We also present a method for determining the optimum profitable semijoin program. The computational complexities of finding the optimum cost semijoin program is high. We present a low cost algorithm which determines a near optimal profitable semijoin program. We do this by converting a semijoin program into a partial order graph. This graph also allows us to maximize the concurrent processing of the semijoins. It is shown that the minimum response time is given by the largest cost path of the partial order graph. We can use this reducibility as a post optimizer for the SDD-1 query optimization algorithm. Finally, it is shown that the least upper bound on the length of any profitable semijoin program is $N * (N - 1)$ for a query graph of N nodes.**

Index Terms—**Distributed databases, full reducer semijoin program, partial order graph, profitable semijoin.**

I. Introduction

An important performance issue in distributed database systems is the implementation of logical relationships of data elements stored across sites. An example of this is the high cost of performing the join of relations stored at different sites. The straightforward approach to implement the join is to send one of the join participating relations to the site of the other relation and perform the join at that site. This requires much movement of data between sites. The objective of join query optimization is to reduce the cost of this inter site data transmission and to move data in parallel so as to minimize the response time.

Several distributed query optimization algorithms have been proposed that minimize the amount of this data transmission [1], [3]-[6], [8], [10]-[12]. Wong has proposed a greedy algorithm that is based on selecting the most profitable semijoin at each step [10]. This has been implemented in SDD-1 [4]. Yao and Hevner [1] have proposed an algorithm which is optimal for a class of queries in which only one attribute is common to all the relations in the join. They have considered optimizing the response time of a query. There are other query optimization strategies for distributed database systems some of which are extensions of centralized query processing [6], [8].

Manuscript received April 28, 1986; revised August 21, 1986.

S. Pramanik is with the Department of Computer Science, Michigan State University, East Lansing, MI 48824.

D. Vineyard with the Department of Computer Science, University of Michigan—Flint, Flint, MI 48502.

IEEE Log Number 8822453.

To minimize the amount of data transmission between sites an approach using semijoin programs has been proposed [4]. Instead of performing joins in one step, semijoins are performed first to reduce the size of the relations. In the next step joins are performed on the reduced relations. Semijoins are also used in performing joins for database machines [2], [7], [9].

In this paper we consider query optimization using semijoins. It has been shown that a class of queries called tree queries, or acyclic queries, can be answered by using a sequence of semijoins called a semijoin program [3]. Cyclic queries, however, require more elaborate data transmission for reduction, and in some cases semijoins cannot reduce the join relations at all. In this paper we first consider optimum cost semijoin programs for acyclic queries and then extend this to compute a low cost profitable semijoin program. Optimum cost semijoin strategies have been studied by Yu [12] and Chiu [3]. Their strategies handle query graphs where at most one attribute is common to two relations. In this paper we derive optimal semijoin programs for acyclic queries where the relations can have more than one common attribute. Finally, we give a low cost algorithm which converts any profitable semijoin program into a partial order graph. This graph enables us to derive a semijoin program with much reduced total cost while at the same time allowing maximum possible parallel execution of the semijoin program.

II. Tree Queries and Semijoin Programs

The *semijoin* [4] of a relation R_j by another relation R_i over a set of common attributes is defined as the projection of the result of taking the equijoin of R_i with R_j over the attributes of R_j. The semijoin of R_j by R_i denoted (i, j) is computed by transmitting the projection of R_i on the common attributes to the site of R_j and then performing the join of this projection with R_j. In evaluating a relational database query, in general many semijoins are required. We call a sequence of semijoins a semijoin program.

In finding an optimal cost semijoin program it is helpful to consider a *query graph*

$G_Q = (V_Q, E_Q)$ where
$V_Q = \{$set of relations referenced by the query $Q\}$

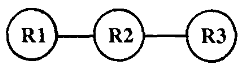


Fig. 1. Query graph.

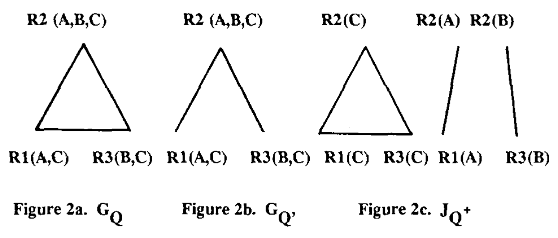


Fig. 2. Equivalent query graphs.

$E_Q = \{(i, j) \mid i \neq j$ and query Q has a join of relations i with $j\}$.

An example of a query graph is given in Fig. 1.

A *query Q is called acyclic* either if its query graph is acyclic or if it is equivalent to a query whose query graph is acyclic. Otherwise, Q is called a cyclic query [3]. For some cyclic query graphs equivalent acyclic query graphs exist. For example, the cyclic query graph G_Q of Fig. 2(a) is reducible to the query graph $G_{Q'}$ of the equivalent query Q' [see Fig. 2(b)]. The algorithm to convert G_Q into $G_{Q'}$ is given below [3].

Step 1: Find the transitive closure graph G_{Q^+} corresponding to the transitive closure Q^+ of Q. Essentially, we are adding edges to G_Q corresponding to the implied joins of the query. In our example

$$G_{Q^+} = G_Q.$$

Step 2: Find the join graph J_{Q^+} for Q^+ by taking the edges from G_{Q^+} over exactly one attribute per component of J_{Q^+}. This is shown in Fig. 2(c).

Step 3: Now construct an acyclic query graph from J_{Q^+} by constructing the spanning forests of J_{Q^+}. If any one of these spanning forests correspond to an acyclic query graph, name that graph $G_{Q'}$.

Please note that step 3 is different from that in Bernstein [3] because we allow more than one attribute in common between two relations. In the rest of this paper we apply our algorithms and theorems to the equivalent acyclic query graph of an acyclic query.

III. Reduced Cover of Semijoin Programs

We denote a *semijoin program* of n semijoins by the sequence $\{(a_i, b_i)\}_{i=1,n}$.

An *embedded chain* is defined to be any subsequence of the semijoin program as follows:

$$\left\{(a_{j_r}, b_{j_r})\right\}_{r=1,m} \text{ where } m <= n,\ r < s \text{ implies } j_r < j_s,$$
$$1 <= j_r <= n, \text{ and } a_{k+1} = b_k.$$

For example, in the semijoin program

$$(1, 2)(1, 3)(4, 5)(3, 6)(5, 7)(8, 9)$$

the following semijoin programs are embedded chains

$$(1, 3)(3, 6) \text{ and } (4, 5)(5, 7).$$

In words, the embedded chain is a subsequence which preserves the order of semijoins in the original semijoin program, and in which any two contiguous semijoins operate on the same relation, the earliest occurring semijoin reduces that relation and the next semijoin uses that reduced relation to reduce another.

We say that a relation R_i is *reduced* by a relation R_j in a semijoin program if the semijoin program has an embedded chain $\{(a_i, b_i)\}_{i=1,m}$ such that $a_1 = R_j$ and $b_m = R_i$.

A relation R_i is said to be *fully reduced* in a query graph if given any relation R_j in the query graph such that $i \neq j$, R_i is reduced by R_j.

A *full reducer program* for a query graph is a semijoin program which reduces each relation in the query graph fully.

An example of a full reducer program for the query graph of Fig. 1 follows:

$$(1, 2), (3, 2), (2, 3), (2, 1).$$

Here R_2 is fully reduced since R_2 is reduced by both R_1 and R_3, similarly, R_1 and R_3 have been fully reduced.

There are many full reducer programs for a given query graph. The number of such programs grows combinatorially with the number of edges of the query graph. For the simple query graph of Fig. 1, there are 28 unique full reducer programs in which each semijoin reduces some relation nontrivially. Our objective is to find the full reducer program of an acyclic query graph which has the least cost. This least cost full reducer program will depend on the cost model, and on the specific data for the relations in the query graph. An algorithm to determine an optimal cost semijoin program for a chain query has been given in [4]. We will propose an algorithm which works for any acyclic query graph. Further, the cost model used can be very general in our algorithm. In fact, this cost model depends only on the size of the relation used to do the reducing. We will apply this model to find a subset of the set of full reducer programs which have less cost than those full reducer programs not in the subset. This subset is considerably smaller than the set of all possible full reducer programs for the query graph in which each semijoin is nontrivial.

In this paper, we concentrate on reducing the communications cost. We assume that the local processing cost has a negligible contribution to the total cost. Thus we need to consider only the cost of transmitting the data. We assume that the transmission cost is given by

$$\text{cost}(n) = c_0 + c_1 * n$$

where n is the amount of data transmitted and c_0 and c_1 are constants [12]. The value of n changes dynamically with the execution of the sequence of semijoins in the semijoin program. The effectiveness of the algorithms given in this paper depends on the accuracy of determining the values of n. Some work to compute the values of n is given in [4], [11].

We will use the following three properties of semijoins to derive the reduced cover for a query graph.

Property 1: Applying the same semijoin twice in succession does not reduce the relation more than a single application of that semijoin. That is $(j, i), (j, i)$ does not reduce i any more than (j, i).

Property 2: Given two successive steps of a semijoin program where a given relation, say i, is always the rightmost (leftmost) of the pair, these steps may be performed in any order with no effect on the cost of the semijoin program.

Property 3: If a semijoin program has two consecutive steps of the type $(k, i), (j, k)$ then the semijoin program found by reversing their order is less expensive if $i \neq j$.

Under Property 2 we can commute the order of performing semijoins without altering the cost of the semijoin program. Property 3 allows us to commute steps in order to derive a less costly semijoin program. In fact, by successively applying the three properties above, we can derive a less costly semijoin program. For example, each step of the following full reducer program for Fig. 1 reduces some relation.

$$(2, 3), (2, 1), (1, 2), (2, 3), (3, 2), (2, 1)$$

By applying the above properties we will derive a semijoin program which has less cost. The above semijoin program is equivalent by property 2 to

$$(2, 1), (2, 3), (1, 2), (2, 3), (3, 2), (2, 1).$$

Now apply property 3 to derive the less expensive program.

$$(2, 1), (1, 2), (2, 3), (2, 3), (3, 2), (2, 1)$$

Apply property 1 to eliminate one of the $(2, 3)$.

$$(2, 1), (1, 2), (2, 3), (3, 2), (2, 1)$$

The above has less cost than the original full reducer semijoin program. From now on we will consider two semijoin programs which can be derived from each other by using property 2 to be the same semijoin program.

The *data state* is defined to be the current values of the tuples of each relation of the database. For one particular data state, one full reducer program may have the minimum cost; but for another data state there may be another full reducer which has less cost.

A reduced cover C_G for the full reducer programs of a query graph G is a set of full reducer programs which has the property that no other full reducer program can have less cost than one of the elements of this set for any data state. Furthermore, no two elements in C_G have the same cost for every data state.

A reduced cover of full reducers for Fig. 1 is given in Fig. 3.

Our objective is to produce an algorithm which will give us a reduced cover for a general query graph. Using this reduced cover will greatly reduce the complexity of the problem of finding the minimal cost full reducer program of a query graph.

1. **(1,2), (3,2), (2,1), (2,3)**
2. **(1,2), (2,3), (3,2), (2,1)**
3. **(3,2), (2,1), (1,2), (2,3)**
4. **(2,1), (1,2), (2,3), (3,2), (2,1)**
5. **(2,3), (3,2), (2,1), (1,2), (2,3).**

Fig. 3. Reduced cover for the query graph of Fig. 1.

IV. Minimal Cover of Single Reducer Programs

A *single reducer program* for relation i of a query graph is a semijoin program in which i is the only relation which is fully reduced. For example:

$$(2, 3), (3, 2), (1, 2)$$

is a single reducer program for relation 2 of the query graph of Fig. 1.

A *prefix* of a semijoin program is the first n steps of a semijoin program. The number n is arbitrarily chosen such that $1 <= n <= m$, the number of steps in the program.

Lemma 1: For any full reducer program there is a unique prefix which is a single reducer program.

Proof: A full reducer program is a finite number of semijoins executed in succession. Each semijoin reduces at most one relation. Thus exactly one of the relations in the query graph is fully reduced first. ■

A *minimal cover* S_G of single reducer programs for a query graph G is a set of single reducer programs which has the property that no other single reducer program can have less cost than one of the elements of this set for any data state. Furthermore, no two elements in this set have the same cost for each data state.

For example, $(1, 2), (3, 2)$ is in S_G but $(2, 3), (3, 2), (1, 2)$ is not. The later is not in S_G although it reduces only R_2 fully. This is because the program $(1, 2), (2, 3)$ is in S_G and it costs less for every data state. Note that $(1, 2), (2, 3)$ reduces node R_3, which is a different node from the original node reduced (R_2).

The theorems and propositions we will discuss depend upon the properties of graphs and trees. We will therefore use the terms node and relation interchangeably.

Theorem 1: A single reducer program for node i in S_G does not reduce node j by i for any j in G.

Proof: We number the nodes of the query graph so that i is now 1 and its children are nodes $2, 3, \cdots, m$. Assume without loss of generality that node 2 is reduced by node 1. Now since node 1 must be reduced by nodes $3, 4, \cdots, m$, by property 3 it is better to perform semijoins $(3, 1), (4, 1), \cdots, (m, 1)$, first. Now node 2 is fully reduced before node 1. This is a contradiction. ■

For example, $(1, 2), (3, 2)$ is in S_G and node 2 does not reduce any node. On the other hand $(2, 3), (3, 2), (1, 2)$ is a single reducer for node 2, but node 2 also reduces node 3. Thus this program is not in S_G.

Lemma 2: A single reducer program in the minimal cover can be extended to a unique full reducer program by the following:

1) Consider the query graph as a tree with the fully reduced node i as the root.

2) For each child, j, of i, reduce j by i.

3) For each subtree with root j, rename j to i and proceed with step 2.

4) Stop when all relations of the query graph have been reduced.

Proof: Given in Bernstein [3]. ■

Theorem 2: There is a one-to-one correspondence between S_G and C_G for a given query graph G.

Proof: Let N be the number of full reducer programs in C_G. Let M be the number of single reducer programs in S_G. $N <= M$ since, by Lemma 1, for each semijoin program in C_G there is a unique prefix which is a single reducer program. This single reducer program is in the minimal cover S_G else by replacing it with a single reducer program of S_G and extending this to a full reducer program by Lemma 2, we have a less costly full reducer program. This is a contradiction since the full reducer program was chosen from C_G.

$M <= N$ since for each single reducer program in S_G, we may append a sequence of steps to create a minimal cost full reducer program by Lemma 2. This full reducer program is in C_G by Lemma 2.

We have shown that $N <= M$ and that $N >= M$, hence $N = M$. ■

The semijoin programs of Fig. 4 form a minimal cover for the query graph of Fig. 1. Note that there is a one to one correspondence between the semijoin programs of Fig. 4, the minimal cover, and the semijoin programs of Fig. 3, the reduced cover. In fact, each semijoin program of Fig. 4 is a prefix of a semijoin program in Fig. 3.

V. Algorithm to Derive Reduced Cover

Define $S_i \subseteq S_G$ to be the set of all single reducers in S_G for the node i and define C_i^j to be the set of full reducer programs in the reduced cover for the subtree of G with root j, where G is considered to be rooted at i.

Proposition 1:

$$|S_i| = \prod_{j \in \{\text{child}(i)\}} |C_j^i|.$$

Proof: By Theorem 1, the node i must be fully reduced before any other node is reduced by i. In particular, each child j of i must be fully reduced in the subtree of which j is the root before i is reduced by j. The semijoin programs for the subtrees are independent, and since there is only one way to reduce i by j once j is reduced, the number of ways to reduce i is the product of the number of ways to reduce the nodes j in their various subtrees. ■

By Theorem 2 we know that

$$|C_G| = \sum_{i \in V} |S_i|.$$

Thus using Proposition 1 we have that

$$|C_G| = \sum_{i \in V} \left(\prod_{j \in \{\text{child}(i)\}} |C_j^i| \right).$$

1. (1,2), (3,2)
2. (1,2), (2,3)
3. (3,2), (2,1)
4. (2,1), (1,2), (2,3)
5. (2,3), (3,2), (2,1)

Fig. 4. Minimal cover of single reducer programs of the query graph of Fig. 1.

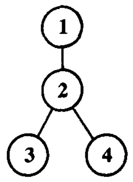


Fig. 5. Query graph.

The above is the formula for computing the number of semijoin programs in C_G. We can derive the semijoin programs in C_G similarly. The query graph of Fig. 5 has 16 semijoin programs in C_G, $|S_1| = |S_3| = |S_4| = 5$, $|S_2| = 1$. Elements of C_G associated with S_1 from the query graph of Fig. 5 are:

(3, 2), (4, 2), (2, 1), (1, 2), (2, 3), (2, 4)

(4, 2), (2, 3), (3, 2), (2, 1), (1, 2), (2, 3), (2, 4)

(2, 4), (4, 2), (2, 3), (3, 2), (2, 1),
    (1, 2), (2, 3), (2, 4)

(3, 2), (2, 4), (4, 2), (2, 1), (1, 2), (2, 3), (2, 4)

(2, 3), (3, 2), (2, 4), (4, 2), (2, 1),
    (1, 2), (2, 3), (2, 4).

VI. Minimum Cost Formulas

In this section we derive a formula for the minimal cost full reducer program. We will use a cost function $K(i, G)$ which gives the minimal cost single reducer program in S_i for node i in G. Note that $K(i, G)$ depends on the data state, but we will not make this explicit in the definition for $K(i, G)$.

The cost is a function of a semijoin program and is defined as the sum of the costs of the semijoins in the semijoin program. Under this interpretation, the parameters we consider are the query graph and the node in the graph which is fully reduced.

Proposition 2:

$$K(i, G) = \sum_{j \in \{\text{child}(i)\}} \left(\min_{p \in G_j} \{K(p, G_j) + K'(p, j)\} + K'(j, i) \right)$$

where G is a tree rooted at i, G_j is the subtree of G rooted at j, and $K'(r, s)$ is the cost of the semijoin program to reduce s by r.

Proof: From the proof of Proposition 1, for each j where j is a child of i, take the least cost semijoin program which fully reduces j in G_j, then add the cost to reduce i by j. ■

We will illustrate the process of finding $K(1, G1)$ for the graph $G1$ in Fig. 6. Our cost model will be defined by the cost $c_{i,j}$ for the semijoin (i, j) where i and j are the unreduced relations. Assume that there is a factor $d < 1$ such that if relation i is reduced by k other relations, the size of relation i is d^k times its original size. Make the assumptions $d = 0.6$, $c_{1,2} = 50$, $c_{2,1} = 100$, $c_{2,3} = 150$, $c_{3,2} = 300$, $c_{2,4} = 200$, and $c_{4,2} = 250$.

$$\begin{aligned} K(1, G1) &= \text{Min}\,\{K(2, G2), K(3, G2) \\ &\quad + K'(3, 2), K(4, G2) + K'(4, 2)\} \\ &\quad + K'(2, 1) \\ K(3, G2) &= \text{Min}\,\{K(2, G3), K(4, G3) \\ &\quad + K'(4, 2)\} + K'(2, 3) \\ K(2, G3) &= K(4, G4) + K'(4, 2) \\ &= K'(4, 2) = c_{4,2} = 250 \\ K(4, G3) &= K(2, G5) + K'(2, 4) \\ &= K'(2, 4) = c_{2,4} = 200 \end{aligned}$$

thus

$$K(3, G2) = \text{Min}\,\{250, 200 + 150\} + 90 = 340$$

similarly

$$K(1, G1) = \text{Min}\,\{550, 340 + 108, 420 + 90\} + 36 = 484.$$

Proposition 2 can be used to find $|V|$ least cost single reducer programs in S_G, one for each of the nodes of G. We will use this result to find the minimal cost full reducer program for G, $M(C_G)$. By Lemma 2, once we have a single reducer program in S_i, there is a unique extension which makes this a full reducer program in C_G. Call this extension X_i. Define $K''(X_i)$ to be the cost of this extension.

Proposition 3:

$$M(C_G) = \min_{i \in G} \{K(i, G) + K''(X_i)\}.$$

Proof: By Proposition 2, $K(i, G)$ has minimal cost. $K''(X_i)$ has minimal cost by Lemma 2. Therefore $M(C_G)$ has minimal cost. ■

It is often useful to know the minimal cost to reduce a certain node, say n, fully, independently of the cost of reducing any other node. This is a similar problem to finding $M(C_G)$. Use $M(n, G)$ to represent the minimal cost to reduce node n in G. Use $X_{i,n}$ to represent the extension of any single reducer for node i in S_i to the semijoin program which reduces node n fully. define K''' as the cost of this extension.

Proposition 4:

$$M(n, G) = \min_{i \in G} \{K(i, G) + K'''(X_{i,n})\}.$$

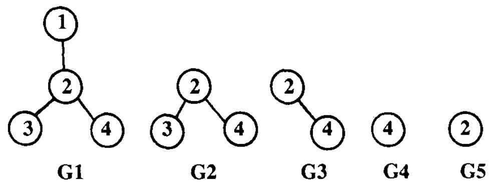


Fig. 6. Query graph and subtrees.

Proof: This follows from Proposition 2 and Lemma 2. ■

VII. Profitable Semijoin Programs

In the previous sections we have considered full reducers, i.e., reducing the relation fully. It may cost more to reduce relations fully than to send the partially reduced relations to the destination site. Thus, we define a profit in performing a semijoin $\sigma = (I, J)$ as

$P(\sigma)$ = (cost of sending the relation J to the destination site) − ((cost of sending the common attributes of relations I and J to the site of J) + (cost of sending the reduced relation J to the destination site)).

Note that $P(\sigma)$ can be negative. We define the profit of a semijoin program $\{\sigma_i\}$ as

$$\sum_i P(\sigma_i).$$

Thus the profit of a semijoin program is the sum of all the profits of all the semijoins taken in the semijoin program. In SDD-1 only profitable semijoins are considered. Note, however, that the semijoin program with optimal overall profit may have locally unprofitable semijoins. In the following section we describe a method for determining the optimal profitable semijoin program.

The profit of a semijoin program can only increase or stay the same if the semijoins are pairwise reordered by using properties 2 and 3. After reordering, some profitable semijoins may become unprofitable. These unprofitable semijoins cannot be removed from the semijoin program because that may reduce the overall profit. For example, assume that all the semijoins in (3, 2), (1, 2), (2, 5), (4, 3) are profitable. The reordering can make (1, 2) unprofitable even though (4, 3), (3, 2), (1, 2), (2, 5) is a more profitable semijoin program. Removing a profitable (or adding an unprofitable) semijoin to the semijoin program may increase the profit of that program. Therefore, we have to consider semijoin programs consisting of all possible combinations of all the semijoins as long as the ordering of the semijoins satisfy properties 2 and 3. Thus in finding the optimum profitable semijoin program we consider the full reducers of C_G because they are already ordered by properties 2 and 3. However, a profitable semijoin program does not have to be a full reducer. Thus we determine the optimum profitable semijoin program by considering all possible subsequences of all the semijoin programs of C_G. The subsequence with the highest profit is the optimum profitable semijoin program. Some of the subsequences in a semijoin program do not have the right ordering according to properties 2 and 3. But equivalent subsequences with the right order exist in another semijoin program of the C_G.

The cost of finding the optimal profitable semijoin program is high. However, there are inexpensive algorithms to find a suboptimal profitable semijoin program such as that in SDD-1. Now we will transform any profitable semijoin program into a more profitable semijoin program.

Proposition 5: If there exist semijoins (a_i, b_i) and (a_j, b_j) in a semijoin program such that $b_j = a_i$ and $i < j$, then moving semijoin (a_j, b_j) before (a_i, b_i) in the semijoin program will reduce cost of (a_i, b_i).

Proof: The proposition follows from the fact that (a_j, b_j) is a profitable semijoin and therefore reduces node a_i. ■

Note that the cost of (a_j, b_j) in the new semijoin program may be greater than in the old semijoin program. For example, if there is some node (a_k, b_k) in the old semijoin program with $b_k = a_j$ and $i < k < j$ then (a_j, b_j) is reduced in the old semijoin program by (a_k, b_k) but not in the new semijoin program.

Theorem 3: If semijoins (a_i, b_i) and (a_j, b_j) are not part of the same embedded chain of the sequence of semijoins $\{(a_k, b_k)\}$, $i <= k <= j$, then semijoin (a_j, b_j) and all the embedded chains of which (a_j, b_j) is the trailing semijoin may be moved before (a_i, b_i) without increasing the cost of any semijoin in the semijoin program.

Proof: By proposition 5 the cost of (a_i, b_i) is reduced. Now assume that the cost of some semijoin is increased. Then that semijoin must be performed before a semijoin which had previously reduced its cost. Thus an embedded chain has been broken. This is a contradiction since the only semijoin which has explicitly been caused to be performed after the embedded chain is (a_i, b_i) and this semijoin is by assumption not part of the embedded chain. ■

We see that the cost of (a_i, b_i) is reduced if (a_j, b_j) is moved before (a_i, b_i) by Theorem 3. By applying Theorem 3 iteratively to a semijoin program, we will reduce the cost further. We will formalize this procedure in Algorithm 1.

Algorithm 1 below derives a new semijoin program from a given program such that the overall profit is increased. The method used is a combination of rearranging of semijoins using Theorem 3 and the addition of profitable semijoins. It uses a partial order graph P to describe the order of performing semijoins. The nodes of P are the semijoins of the semijoin program. The directed edges of P determine the order of semijoins in the derived program. We will use the idea of level of a node for P. Level 0 nodes are source nodes of P. A node (i_k, j_k) which is the terminating node of some edge is at the level Max $\{$level of node (i_r, j_r) such that $(i_r, j_r)(i_k, j_k)$ is an edge of $P\} + 1$.

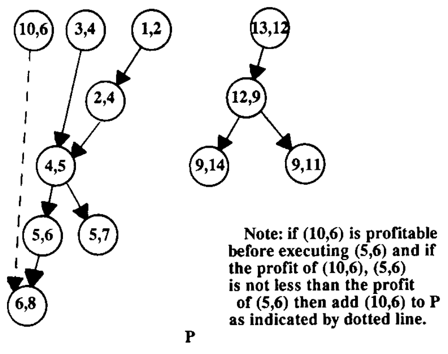
Algorithm 1:

```
Step 1:      [Initialize $P$]. Make $P$ the null graph.
Step 2:      [Add semijoin nodes to $P$]. For each semijoin $(i_k, j_k)$ in the semijoin program order do:
Step 2.a:    If $P$ is the null graph, then $P$ becomes the graph of one node, $(i_k, j_k)$.
Step 2.b:    If $P$ is not null then do:
Step 2.b.1:  For each node $(i_r, j_r)$ in $P$ such that $j_r = i_k$, if an edge from $(i_r, j_r)$ to $(i_l, j_l)$ is not in $P$ where $(i_l, j_l)$ is a previous occurrence of semijoin $(i_k, j_k)$ in $P$, then form the edge in $P$ from $(i_r, j_r)$ to $(i_k, j_k)$.
Step 2.b.2:  For each node $(i_r, j_r)$ in $P$ such that $i_r = j_k$, if there is no path from $(i_r, j_r)$ to $(i_k, j_k)$, form the edge from $(i_k, j_k)$ to $(i_r, j_r)$ in $P$.
Step 2.b.3:  If no semijoins $(i_r, j_r)$ as described in Steps 2.b.1 and 2.b.2 exist in $P$ then make $(i_k, j_k)$ a separate component of $P$.
Step 3:      [Add new semijoins to $P$]. For each node $(i_k, j_k)$ in $P$ do the following:
Step 3.a:    [Find semijoin]. Consider a semijoin $(i_r, j_r)$ from the original query graph which does not appear as the head of an edge in $P$ to $(i_k, j_k)$, but in which $j_r = i_k$. Consider the node $i_k$ from the query graph as having been reduced by all semijoins above $(i_k, j_k)$ in $P$ except for those semijoins forming edges $(i_n, j_n)(i_k, j_k)$ in $P$. If semijoin $(i_r, j_r)$ is not profitable go to step 3.d.
Step 3.b:    [Add]. Let $\sigma = \Sigma$ Profit$((i_n, j_n))$ where $(i_n, j_n)(i_k, j_k)$ is an edge in $P$. Let $\sigma' = \Sigma$ Profit$((i_n, j_n))$ where $(i_n, j_n)(i_k, j_k)$ is an edge in $P$ or $(i_n, j_n) = (i_r, j_r)$. If $\sigma' >= \sigma$ then add $(i_r, j_r)$ to $P$ and form the edge $(i_r, j_r)(i_k, j_k)$.
Step 3.c:    [Attach]. For any node $(i_m, j_m)$ in $P$ such that $j_m = i_r$, if there is no edge from $(i_m, j_m)$ to $(i_s, j_s)$ where $(i_s, j_s)$ is another occurrence of semijoin $(i_r, j_r)$ then form the edge $(i_m, j_m)(i_r, j_r)$. Now if a cycle has been formed in $P$, delete this edge.
Step 3.d:    Continue Step 3.a with another $(i_r, j_r)$ if one exists. Else continue Step 3.a with another node $(i_k, j_k)$ in $P$.
Step 3.e:    Halt when no profitable semijoins $(i_r, j_r)$ exist.
Step 4:      Write the derived semijoin program from the graph $P$ in the following manner: For level 0 to Max $\{$level(node) | node $\in P\}$ list the nodes of $P$ in level order.
```

In other words, Algorithm 1 is used to determine the embedded chains found by reordering the semijoins of the

Semijoin Program:

(4,5),(6,8),(5,6),(9,14),(1,2),(13,12),(3,4),(12,9),(2,4),(9,11),(5,7)


10,6
3,4
1,2
13,12
2,4
12,9
4,5
9,14
9,11
5,6
5,7
6,8
Note: if (10,6) is profitable before executing (5,6) and if the profit of (10,6), (5,6) is not less than the profit of (5,6) then add (10,6) to P as indicated by dotted line.
P

Derived Semijoin Program:

(1,2),(3,4),[(10,6)],(13,12),(2,4)(12,9),(4,5),(9,14),(9,11),(5,6),(5,7),(6,8)

Fig. 7. Semijoin program and partial order graph P.

original semijoin program. Step 2 places the semijoins in the graph P so that embedded chains are formed, but in a manner which prevents cycles in P from being created. Step 3 is used to place semijoins in P which are profitable in the data state corresponding to P but which were not in the original semijoin program. Fig. 7 below gives an example of a derived semijoin program.

With modifications, this algorithm can be used to generate a semijoin program. Essentially, use some method to choose a profitable semijoin to add to the new program (instead of taking the semijoin from a given semijoin program) and add this semijoin to the graph P. One method would be to choose the locally most profitable semijoin as in SDD-1 to add to the graph.

Proposition 6: Algorithm 1 gives a derived semijoin program which is at least as profitable as the original program. That is, Algorithm 1 has the inclusion property.

Proof: Step 2 corresponds to rearranging the semijoins according to properties 2 and 3 and can not decrease the profitability by Theorem 3. Step 3 guarantees that the global profit will not decrease. Therefore, the derived semijoin program is at least as profitable as the original program. ■

Note that as a result of step 2 we may have locally unprofitable semijoins in the semijoin program. These semijoins were part of a profitable semijoin program before applying step 2 and are now part of a more profitable program. Step 3 is used to place semijoins in the program which were made unprofitable by the inclusion of another semijoin in the program. This only occurs if the semijoin placed in the program by step 3 does not reduce the profitability of the already occuring semijoins significantly.

It is quite possible for a hill climbing algorithm for semijoin programs to create a semijoin program which violates properties 1, 2, or 3. These same algorithms can also, by order of inclusion, prevent profitable semijoins from being included in the semijoin program. Our algorithm will give us a better, although not the optimal in many cases, semijoin program.

Proposition 7: Algorithm 1 will stop after a finite number of steps.

Proof: By Proposition 11 we know that the number of semijoins is bounded in a semijoin program in which each semijoin reduces some relation. Algorithm 1 adds nodes to P, each corresponding to a semijoin which reduces some relation. Therefore, the number of steps executed by Algorithm 1 is bounded. ■

Proposition 8: Let N be the maximum number of semijoins possible for a given query Q. Then the worst case computational complexity of Algorithm 1 applied to a semijoin program for Q is $O(N^2)$.

Proof: Algorithm 1 consists of placing the semijoins on P as nodes. For each of the at most N semijoins, this involves a scan of P. Therefore, for the worst case, Algorithm 1 has complexity $O(N^2)$. ■

VIII. Concurrency in a Semijoin Program

A semijoin program is a linear structure, it implies a linear processing of the semijoins. The graph P is a two-dimensional structure. If we consider the graph P to be a data flow graph, then P is a schedule of the concurrent semijoin processing of the semijoin program. In fact, the graph P gives the maximum concurrency allowed in a given semijoin program while achieving the best reduction in the total cost.

Given a semijoin program, the partial order graph P gives the minimal cost of performing these semijoins. For this minimal cost semijoin program P provides the minimal response time. The minimal cost comes from increasing the size of embedded chains. The minimal response comes from executing these semijoins concurrently.

The minimal response time is given by the maximal cost path from a source node to a sink node of the graph, assuming that the response time is directly proportional to the cost. This follows directly from the properties of data flow graphs.

IX. Longest Profitable Semijoin Program

There are algorithms which determine the low cost semijoin program by choosing locally optimal profitable semijoins. These algorithms may select a semijoin program which is not in the set of optimal profitable semijoin programs. For example, using Fig. 1, (2, 1)(3, 2), (2, 3), (1, 2), (2, 1), (2, 3) is not in the optimal set, but each semijoin in the program reduces a node and thus can be profitable. This is the longest such semijoin program for Fig. 1. The following propositions give the longest profitable semijoin programs for a given query graph.

Proposition 9: An upper bound on the number of profitable semijoins in a semijoin program for an acyclic query graph of N nodes is $N * (N - 1)$.

Proof: Consider any semijoin (A, B) in a semijoin program. If (A, B) is profitable then B must be reduced by at least one node. Obviously, if each semijoin reduces its terminal node by exactly one node, this is the worst case for minimizing the number of semijoins. If each semijoin reduces its terminal node by one node, it takes $N - 1$ semijoins to fully reduce any given node. Therefore, $N * (N - 1)$ is an upper bound on the number of profitable semijoins for a query graph of N nodes. ■

Proposition 10: There is a semijoin program which takes $N * (N - 1)$ profitable semijoins for an acyclic query graph of N nodes.

Proof: The proof is by induction on the number of nodes of the query graph.

Basis: For $N = 2$ with nodes A and B, the semijoins (A, B), (B, A) reduce the graph fully.

Induction Hypothesis: Assume that the proposition is true for $N = K$.

Induction Step: Consider an acyclic query graph with $N = K + 1$ nodes. Consider any leaf node of the graph, call it B and call its parent node A. Remove B from the graph, then by our hypothesis there is a semijoin program of $K * (K - 1)$ nodes for the resulting K node graph. Extend this program to the entire graph as follows:

Step 1: (A, B) is the first semijoin in the new program.

Step 2: After any semijoins (X, A) insert the semijoin (A, B)

Step 3: At the end of the old program insert the semijoins (B, A) followed by all (A, X), followed by all (X, Y), etc. until all nodes have been reduced by B.

The number of semijoins in this extended program is $(K + 1) * K$. ■

Proposition 11: The least upper bound on the number of profitable semijoins in a semijoin program for a graph of N nodes is $N * (N - 1)$.

Proof: Follows from Propositions 9 and 10. ■

X. Conclusion

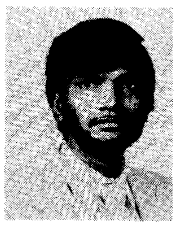
An algorithm to find the minimal cost full reducer and single reducer semijoin programs is given. The computational complexities of this algorithm is high. This algorithm has lead to the development of a better heuristic to compute the near optimal profitable semijoin program. The partial order graph used in this heuristic enables us to exploit the parallel execution of the semijoins in the program.

Acknowledgment

The authors would like to thank the reviewers for their helpful comments.

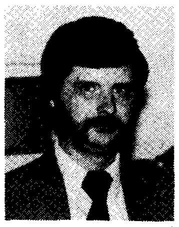
References

[1] P. Apers, A. Hevner, and S. B. Yao, "Optimization algorithms for distributed queries," *IEEE Trans. Software Engineering*, vol. SE-9, no. 1, pp. 57-68, Jan. 1983.
[2] E. Babb, "Implementing a relational database by means of specialized hardware," *ACM Trans. Database Syst.*, vol. 4, no. 1, pp. 1-29, Mar. 1979.
[3] P. Bernstein and D. Chiu, "Using semijoins to solve relational queries," *J. ACM*, vol. 28, no. 1, pp. 25-40, Jan. 1981.
[4] P. Bernstein, N. Goodman, E. Wong, C. Reeve, and J. Rothnie, "Query processing in a system for distributed databases (SDD-1)," *ACM Trans. Database Syst.*, vol. 6, no. 4, pp. 602-625, Dec. 1981.
[5] D. Chiu and Y. Ho, "A methodology for interpreting tree queries into optimal semi-join expressions," in *Proc. ACM SIGMOD*, May 1980, pp. 169-178.
[6] R. Epstein, M. Stonebraker, and E. Wong, "Distributed query processing in a relational database system," in *Proc. ACM SIGMOD*, May 1978, pp. 169-180.
[7] S. Pramanik and F. Fotouhi, "An index database machine—An efficient m-way join processor," *The Comput. J.*, vol. 29, no. 5, pp. 430-445, Oct. 1986.
[8] M. Stonebraker and E. Neuhold, "A distributed database version of INGRESS," in *Proc. Second Berkeley Workshop Dist. Data Management and Computer Networks*, 1977, pp. 19-36.
[9] S. Su, L. Nguyen, A. Emam, and G. Lipovskky, "The architectural features and implementation techniques of the multicell CASSM," *IEEE Trans. Comput.*, vol. C-28, no. 6, pp. 430-445, June 1979.
[10] E. Wong, "Retrieving dispersed data from SDD-1: A system of distributed databases," in *Proc. Second Berkeley Workshop Dist. Data Management and Computer Networks*, 1977, pp. 217-235.
[11] C. Yu and C. Chang, "Distributed query processing," *ACM Comput. Surveys*, vol. 16, no. 4, pp. 399-433, Dec. 1984.
[12] C. Yu, Z. Ozsoyoglu, and K. Lam, "Optimization of distributed tree queries," *J. Comput. Syst. Sci.*, vol. 29, no. 3, pp. 409-445, Dec. 1984.



Sakti Pramanik received the B.S. degree in electrical engineering from the Calcutta University, Calcutta, India, where he received the best student award and two gold medals. He received the M.S. degree in electrical engineering from the University of Alberta, Edmonton, Canada, and the Ph.D. degree in computer science from the Yale University, New Haven, CT.

He was with Bell Laboratories, Naperville, IL, where he was involved with the design and development of software architectures of ESS systems. Currently he is a faculty of computer science at Michigan State University, East Lansing. His research interests include query optimization, main memory databases, parallel processing architectures, and computational complexity.



David Vineyard received the B.A. degree in mathematics and the B.S. degree in computer science from the University of Michigan—Flint, and the M.A. degree in mathematics from the University of Michigan, Ann Arbor.

He has been a Lecturer in computer science for the University of Michigan—Flint since 1981 and is currently a Ph.D. student in computer science at Michigan State University, East Lansing.