Distributed Query Optimization by One-Shot Fixed-Precision Semi-Join Execution

Chihping Wang
Department of Computer Science
University of California
Riverside, CA 92521-0135
E-mail: wang@cs.ucr.edu

Victor O.K. Li*
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089-0272
E-mail: vli@irving.usc.edu

Arbee L.P. Chen†
Department of Computer Science
National Tsing Hua University
Taiwan, ROC
E-mail: alpchen@cs.nthu.edu.tw

Abstract

We propose a new semi-join execution strategy which allows parallelism and processes multiple semi-joins simultaneously. The query optimization problem based on this new strategy is NP-Hard. Nevertheless, in practice most of the parameters needed for query optimization, such as relation cardinality and selectivity, are of fixed-precision. Imposing this fixed-precision constraint, we develop an efficient distributed query processing algorithm. For situations where the fixed-precision constraint does not apply, we propose a method to truncate the parameters and use the same algorithm to find near-optimal solutions. By analyzing the truncation errors, we provide a quantitative comparison between the near-optimal solutions and the optimal ones.

1 INTRODUCTION

Query processing in distributed relational databases [6] often requires shipping relations between different sites. To reduce the data transmission cost, *semi-joins* were introduced [2,3]. A semi-join from relation R_i to relation R_j, denoted by $R_i \rightarrow R_j$, is defined as $\prod_{R_j}(R_i \leftrightarrow R_j)$, where $R_i \leftrightarrow R_j$ is the join of R_i and R_j, and $\prod_A(B)$ the projection of relation B on the attributes of relation A. In a distributed database system, it is implemented as follows: Project R_i on the join attributes (of the join between R_i and R_j), then ship this projection (called a *semi-join projection*) to the site of R_j and perform the join with R_j.

It has been proposed that a distributed query be processed as follows [1,4]:

1. **Initial local processing**: all local operations including selections and projections are processed.
2. **Semi-join processing**: the only operations left after initial local processing are joins between relations at different sites. A *semi-join program* is derived from the remaining join operations and executed to reduce the size of the relations.
3. **Final Processing**: all relations which are needed to calculate the answer of the query are transmitted to a final site where joins are performed and the answer to the query obtained.

Numerous algorithms [1,7,8,9,10,11,15,16,17] have been developed to determine a semi-join program for optimal distributed query processing. Most semi-join algorithms favor executing semi-joins sequentially such that the reduction effect of a semi-join can be propagated to reduce the cost of later semi-joins. For example, the cost of $R_i \rightarrow R_j$ may be lowered if another semi-join $R_k \rightarrow R_i$ is executed first. However, performance studies [5,12] show that such semi-join processing strategies are sometimes inefficient for the following reasons:

1. **Loss of parallelism**: The sequential execution of semi-joins excludes the possibility of parallel semi-join execution in a distributed system.
2. **Processing overhead**: Before $R_j \rightarrow R_i$ is executed, R_j has to be scanned in order to generate the semi-join projection. If $R_j \rightarrow R_k$ also appears in the sequential semi-join program, R_j has to be scanned again, which increases the processing overhead.
3. **Loss of global semi-join optimization**: The sequential execution of semi-joins excludes the possibility of performing multiple semi-joins to the same relation simultaneously, for which global optimization techniques [14] may be applied.

*This work was supported in part by the United States department of Defense Joint Services Electronics Program under Contract No. F49620-88-C-0067.

†This work was supported in part by the Republic of China National Science Council under Contract No. NSC 80-0408-E-007-01.

4. **Inaccurate semi-join reduction estimation**: In order to find a good processing strategy, it is needed to accurately estimate the cost and reduction benefit of semi-joins. If such estimation is done each time after a semi-join is executed, too much processing cost may be incurred. If all such estimates are done before the semi-join processing, the accuracy may be low because estimation errors may propagate and be magnified through the sequential execution of semi-joins. This inaccuracy affects the semi-join algorithm's ability to determine an optimal strategy.

To alleviate the above problems, we propose a new semi-join processing procedure, named ***one-shot semi-join execution***. This method executes all applicable semi-joins to the relations at a time. That is, each relation will be reduced by a set of semi-joins at a time, and the semi-join processing at all sites can be performed simultaneously. As a result, each relation needs to be scanned only once to process all applicable semi-joins. These semi-joins can be processed employing a global optimization algorithm. Moreover, since all applicable semi-joins are executed at one shot, no inaccurate estimation of the semi-join cost and benefit will be propagated. The query optimizer therefore decides a semi-join program which is a set instead of a sequence of semi-joins.

In this paper, we consider using one-shot semi-join execution to optimize the transmission cost of distributed query processing. This problem has been shown to be NP-Hard [13], which suggests that some restrictions have to be imposed before an efficient algorithm can be developed. We observe that in most situations, the cardinality of relations, selectivities, and other parameters relevant to the semi-join processing are collected by the system through statistics or estimates. As a result, these parameters are usually of fixed precision. With the fixed-precision constraint, we develop a polynomial-time algorithm to solve the fixed-precision one-shot semi-join optimization problem. We analyze this algorithm and study how good a heuristic it will be if the fixed-precision constraint is relaxed.

The rest of the paper is organized as follows. In section 2, we describe one-shot semi-join execution. In section 3, we consider applying one-shot semi-join execution on a single relation and formulate it into a mathematical programming problem. A polynomial-time fixed-precision algorithm is developed in section 4. In section 5, we extend our results to general queries. In section 6, we propose to truncate parameters and apply our algorithm as a heuristic when the fixed-precision assumption does not apply. We show that by carefully choosing the truncation and hence the precision, the semi-join programs derived by this heuristic can be arbitrarily close to the optimal one. We conclude this paper in section 7.

2 ONE-SHOT SEMI-JOIN EXECUTION

As described in the last section, the goal of one-shot semi-join execution is to remedy the inefficiency of traditional semi-join processing algorithms, which favor sequential execution of semi-joins. Under this new method, the initial local processing and final join processing of a distributed query remain the same. However, the query optimizer has to decide a set of semi-joins to be executed first and the semi-join processing step is further partitioned into three phases, namely, the projection phase, the transmission phase, and the reduction phase. They are explained in the following:

THE PROJECTION PHASE During the projection phase, a relation R_i is scanned once to generate all the necessary semi-join projections. That is, if $R_i \rightarrow R_{j_1}$, $R_i \rightarrow R_{j_2}$, $\cdots$, $R_i \rightarrow R_{j_k}$, are to be executed, R_i is scanned once to generate $\prod_{r_{j_1}} R_i$, $\prod_{r_{j_2}} R_i$, $\cdots$, $\prod_{r_{j_k}} R_i$, where r_{j_k} is the join attributes between R_{j_h} and R_i. On the other hand, a traditional semi-join algorithm may decide to perform a sequence of semi-joins, e.g., $R_i \rightarrow R_{j_1}$, followed by $R_{j_1} \rightarrow R_i$, followed by $R_i \rightarrow R_{j_2}$. $\prod_{r_{j_2}} R_i$ cannot be generated at the same time as $\prod_{r_{j_1}} R_i$ is generated because R_i has to be reduced by $R_{j_1} \rightarrow R_i$ first.

THE TRANSMISSION PHASE All the semi-join projections are then transmitted in parallel to the corresponding sites to perform semi-joins.

THE REDUCTION PHASE After the transmission phase, for each semi-join to a relation R_i, its semi-join projection is available at the site where R_i resides. Therefore, global semi-join optimization is possible. One possible strategy is to hash each semi-join projection first, then scan R_i once to process these semi-joins. Each tuple in R_i is checked against the semi-join projections by using hashing. A tuple of R_i appears in the result only if it finds a matching value from each semi-join projection on the join attributes. Global semi-join optimization is impossible if semi-joins are executed sequentially as in many existing semi-join algorithms.

3 REDUCING A SINGLE RELATION BY ONE-SHOT SEMI-JOINS

In this section, we study the problem of using one-shot semi-join execution to reduce a single relation. Our results will be extended to the optimization of general queries.

Suppose a relation R_0 has to be sent to the final processing site to join with other relations. To reduce the transmission cost of sending R_0, we may apply a one-shot semi-join execution on R_0 first. A one-shot semi-join execution consists of a number of semi-joins which can be executed without waiting for the completion of other semi-joins. All semi-joins of the form $R_i \rightarrow R_0$ are candidates to be included in the execution. Each of these semi-joins reduces the size of R_0 but also incurs extra transmission cost. The problem is to decide which semi-joins should be included in the one-shot semi-join execution in order to minimize the overall transmission cost of sending R_0.

We will next describe the semi-join reduction model and the transmission cost model. They will be used to formulate the single-relation one-shot semi-join optimization problem.

3.1 THE SEMI-JOIN REDUCTION MODEL

A selectivity model [3] has been developed to predict the reduction effect of semi-joins. Under this model, we may assume that associated with each semi-join $R_i \rightarrow R_j$ is a rational number ρ_i^j, ranging from 0 to 1. ρ_i^j is called the selectivity of $R_i \rightarrow R_j$. After $R_i \rightarrow R_j$ is executed, the size of R_j becomes $\rho_i^j \cdot \mid R_j \mid$, where $\mid R_j \mid$ denotes the original size of R_j. We further assume that after the one-shot execution of a set of semi-joins $\{ R_i \rightarrow R_j \mid i \in S\}$, the size of R_j becomes $(\prod_{i\in S} \rho_i^j) \cdot \mid R_j \mid$. We use ρ_i to denote ρ_i^0.

3.2 THE TRANSMISSION COST MODEL

The transmission cost of $R_i \rightarrow R_j$ is denoted by u_i^j. We use u_i to denote u_i^0. We assume the transmission cost of sending R_j to the final processing site is $C_j \cdot X + D_j$, where C_j, D_j, are positive constants and X is the size of R_j after the one-shot semi-join execution.

3.3 FORMULATION

Suppose $U = \{ R_i \rightarrow R_0 \mid 1 \le i \le n\}$ is the set of semi-joins which can be applied to reduce the size of R_0. Consider the one-shot execution of a subset of U, i.e., $\{ R_i \rightarrow R_0 \mid i \in B\}$ where $B \subset \{1, 2, \cdots, n\}$. Our goal is to find B such that the total transmission cost is minimized. The transmission cost incurred by the semi-joins is $\sum_{i\in B} u_i$. The transmission cost of sending R_0 after the semi-join reduction is $C_0 \cdot (\prod_{i \in B} \rho_i) \cdot \mid R_0 \mid + D_0$. The total transmission cost is therefore $(\sum_{i\in B} u_i) + (C_0 \cdot \prod_{i \in B} \rho_i \cdot \mid R_0 \mid) + D_0$. If we define $s_i = \frac{u_i}{C_0 \cdot \mid R_0 \mid}$, optimizing the above objective function is equivalent to minimizing the following:

$$SP(B) \quad = \quad \sum_{i\in B} s_i + \prod_{i\in B} \rho_i \qquad (1)$$

Notice D_0 is dropped from our objective function because it is constant with respect to B.

The total transmission cost corresponding to B, TS(B), can be expressed as a function of SP(B) in the following:

$$TS(B) \quad = \quad C_0 \cdot \mid R_0 \mid \cdot SP(B) + D_0 \qquad (2)$$

Notice that $TS(\emptyset) = C_0 \cdot \mid R_0 \mid \cdot SP(\emptyset) + D_0$, which corresponds to the one-shot execution strategy where no semi-joins are performed and R_0 is not reduced. Therefore, $SP(\emptyset) = 1$.

We use SUM(B) to denote $\sum_{i\in B} s_i$, and PROD(B) to denote $\prod_{i\in B} \rho_i$. We say a set B is *optimal* if it minimizes the objective function SP(B).

Next we study the constraints on s_i and ρ_i. $s_i \cdot C_0 \cdot \mid R_0 \mid$ represents the transmission cost. Therefore, $s_i > 0$. Since ρ_i is a selectivity, $0 < \rho_i \le 1$. Furthermore, any optimal set B cannot have an element i with $s_i > 1$; else SP(B)> SP($\emptyset$) and B is not optimal. Consequently, only those i's with $0 < \rho_i, s_i \le 1$ can be included in the optimal set.

To summarize the discussion above, we define a mathematical programming problem, called Sum Product Optimization (SPO) as follows:

Definition 1 *(SPO)*
Given $F = \{ (s_1, \rho_1), (s_2, \rho_2), \cdots, (s_n, \rho_n)\}$, where $s_i, \rho_i \in Q^{(0,1]}$ (rational in (0,1]), find $B \subset \{1, 2, \cdots, n\}$ such that SP(B) is minimum.

One can determine the minimum of SP(B) by examining all possible B's. However, this is inefficient because there are 2^n possible B's. Actually, SPO has been shown to be NP-Hard [13].

4 A POLYNOMIAL-TIME ALGORITHM FOR THE FIXED-PRECISION SPO

Despite the fact that SPO is an inherently difficult problem, a polynomial-time algorithm exists if each s_i

is represented in binary only to the Kth position after the binary point. That is, the representation of s_i is $\sigma_i \cdot 2^{-K}$ where σ_i is an integer between 0 and 2^K. Here K is called the precision of the problem. This constraint limits the number of possible values $\sum_{i\in B} s_i$ can take. Thus a recursive relation can be defined to minimize SP(B). This recursive relation is then used to derive a polynomial-time dynamic programming algorithm.

4.1 THE RECURSIVE RELATION

Consider an instance of SPO, namely F = { (s_1, ρ_1), (s_2, ρ_2), ⋯, (s_n, ρ_n)}. Using $A_i(r)$ to represent the minimum value of SP(B) over all possible B ⊂ {1, 2, ⋯, i} with the constraint that $\sum_{k\in B} s_k = r$, we obtain

$$A_i(r) = \begin{cases} 1 & \text{if } i=r=0 \\ \infty & \text{if } i=0, r\neq 0 \\ \min\{A_{i-1}(r), TMP_i(r)\} & \text{otherwise} \end{cases}$$

where

$$TMP_i(r) = \begin{cases} \infty & \text{if } A_{i-1}(r-s_i) \\ & \quad = \infty \\ r + \rho_i \cdot (A_{i-1}(r-s_i) & \\ \quad -(r-s_i)) & \text{otherwise} \end{cases}$$

In the above definition, $A_i(r) = \infty$ if there is no B ⊂ {1, 2, ⋯, i} such that $\sum_{k\in B} s_k$ = r. $TMP_i(r)$ can be explained as follows: Let B be a subset of $\{1,2,\cdots,i-1\}$ such that $SUM(B) = r-s_i$ and $SP(B) = A_{i-1}(r-s_i)$. Then $A_{i-1}(r-s_i) - (r-s_i) = PROD(B)$ and $TMP_i(r) = SP(B\bigcup\{i\})$. Notice $B\bigcup\{i\}$ is a subset of $\{1,2,\cdots,i\}$ and $SUM(B\bigcup\{i\}) = r$.

$A_i(r)$ is defined over $0 \le i \le n$ and $0 \le r \le \sum_{i=1}^{n} s_i$. Clearly, $A_i()$ can be derived from $A_{i-1}()$. For $0 < i \le n$, let B be the set corresponding to $A_i(r)$, namely, B ⊂ {1, 2, ⋯, i} and SP(B) = $A_i(r)$. There are two cases: (1) i ∉ B, then $A_i(r) = A_{i-1}(r)$; (2) i ∈ B, then B - {i} corresponds to $A_{i-1}(r-s_i)$ and $A_i(r) = TMP_i(r)$.

The minimum value of SP(B) over all B ⊂ {1, 2, ⋯, n} is $min_{\forall r} A_n(r)$

To derive an efficient dynamic programming algorithm to compute $min_{\forall r} A_n(r)$, we may redefine $A_i(r)$ according to the following definition and theorem.

Definition 2 *$B \subset \{1,2,\cdots,n\}$ is strict if and only if $\forall\ B' \subset B,\ B' \neq \emptyset \Rightarrow SP(B') < 1$.*

Theorem 1 *There exists an optimal B which is strict.*

Proof:
It is enough to show that if B is optimal but not strict, we can always find a proper subset of B, say B′, which is also optimal.

Suppose B is optimal and there exists non-empty B″ ⊂ B, SUM(B″) + PROD(B″) ≥ 1. We claim B′ = B - B″ is also optimal.

SP(B) = (SUM(B′) + SUM(B″)) + (PROD(B′)· PROD(B″)). Therefore,

$$\begin{aligned} & SP(B) - SP(B') \\ = \ & SUM(B'') - PROD(B')\cdot(1 - PROD(B'')) \\ \ge \ & SUM(B'') - PROD(B')\cdot SUM(B'') \\ = \ & SUM(B'')\cdot(1 - PROD(B')) \end{aligned}$$

The last line is non-negative because $\forall i,\ 0 < \rho_i \le 1$.

Q.E.D.

From Theorem 1, one can restrict the search for the optimal B to strict sets, i.e., $0 < A_i(r) \le 1$. Since $r < A_i(r)$, we can also restrict r to $0 \le r < 1$. We redefine $A_i(r)$ as:

$$A_i(r) = \begin{cases} 1 & \text{if } i=r=0 \\ \infty & \text{if } i=0, r\neq 0 \\ min\{A_{i-1}(r), TMP_i(r)\} & i>0 \text{ and} \\ & TMP_i(r) < 1 \\ A_{i-1}(r) & \text{otherwise} \end{cases}$$

We use $C_i(r)$ to record a subset of {1, 2, ⋯, i} such that $SP(C_i(r)) = A_i(r)$. If $A_i(r) = A_{i-1}(r)$, then i should not be included in $C_i(r)$ and $C_i(r) = C_{i-1}(r)$. If $A_i(r) = TMP_i(r) \neq A_{i-1}(r)$, however, $C_i(r)$ can be constructed by including i in $C_{i-1}(r-s_i)$. A recursive relation on $C_i(r)$ follows:

$$C_i(r) = \begin{cases} \emptyset & \text{i= 0 and r=0} \\ C_{i-1}(r) & i > 0 \text{ and } A_i(r) \\ & = A_{i-1}(r) \\ C_{i-1}(r-s_i)\bigcup\{i\} & i > 0 \text{ and } A_i(r) \\ & = TMP_i(r) \neq A_{i-1}(r) \\ undefined & \text{otherwise} \end{cases}$$

4.2 CORRECTNESS

We shall show that $min_{0\le r<1} A_n(r)$ is the minimum SP() and $SP(C_i(r)) = A_i(r)$.

Lemma 1 *$\forall i,\ 0 \le i \le n,\ \forall r,\ 0 \le r < 1$, if $A_i(r) \neq \infty$, then $C_i(r) \subset \{1, 2, \cdots, i\}$, $SP(C_i(r)) = A_i(r)$, and $SUM(C_i(r)) = r$.*

Proof:
The proof is by induction on i. The hypothesis is true when i = 0.

Suppose the hypothesis is true for i ≤ k. Now consider i ≤ k+1. There are two cases:

1. $\mathbf{A_{k+1}(r) = A_k(r) \neq \infty}$. From induction hypothesis, $C_k(r) \subset \{1, 2, \cdots, k\}$, $SP(C_k(r)) = A_k(r)$, and $SUM(C_k(r)) = r$. By definition $C_{k+1}(r) = C_k(r)$. Therefore, $C_{k+1}(r) \subset \{1, 2, \cdots, k+1\}$, $SP(C_{k+1}(r)) = SP(C_k(r)) = A_k(r) = A_{k+1}(r)$, and $SUM(C_{k+1}(r)) = SUM(C_k(r)) = r$. The hypothesis holds for $i = k+1$.

2. $\mathbf{A_{k+1}(r) = TMP_{k+1}(r) \neq A_k(r)}$. This implies $A_k(r - s_{k+1}) \neq \infty$. From induction hypothesis, $SP(C_k(r-s_{k+1})) = A_k(r-s_{k+1})$, and $SUM(C_k(r-s_{k+1})) = r - s_{k+1}$. Thus $TMP_{k+1}(r) = SUM(C_k(r-s_{k+1})) + s_{k+1} + \rho_{k+1} \cdot (SP(C_k(r - s_{k+1})) - SUM(C_k(r - s_{k+1}))) = SP(C_k(r - s_{k+1}) \bigcup \{k + 1\})$. But $C_{k+1}(r) = C_k(r-s_{k+1}) \bigcup \{k + 1\}$ by definition. Therefore, $SP(C_{k+1}(r)) = TMP_{k+1}(r) = A_{k+1}(r)$, and $SUM(C_{k+1}(r)) = SUM(C_k(r-s_{k+1}) \bigcup \{k + 1\}) = r - s_{k+1} + s_{k+1} = r$. From induction hypothesis, $C_k(r-s_{k+1}) \subset \{1, 2, \cdots, k\}$. Therefore $C_{k+1}(r) = C_k(r-s_{k+1}) \bigcup \{k + 1\} \subset \{1, 2, \cdots, k+1\}$.

Q.E.D.

Lemma 2 *$\forall i$, $0 \leq i \leq n$, $\forall$ strict set $B \subset \{1,2,\cdots,i\}$, $SP(B) \geq A_i(SUM(B))$.*

Proof:
Since B is strict, $0 \leq SUM(B) < 1$. Therefore, $A_i(SUM(B))$ is defined. The hypothesis is true for $i = 0$ because $A_0(SUM(\emptyset)) = 1 = SP(\emptyset)$. Suppose the hypothesis is true for all $i \leq k$. Let $B \subset \{1, 2, \cdots, k+1\}$ and B be strict.

There are two cases:

1. $\mathbf{k+1 \notin B}$. From induction hypothesis $A_k(SUM(B)) \leq SP(B)$. But $\forall i$ and $\forall r$, the definition of $A_{i+1}(r)$ guarantees $A_{i+1}(r) \leq A_i(r)$. Thus $A_{k+1}(SUM(B)) \leq SP(B)$.

2. $\mathbf{k+1 \in B}$. From the definition of strict set, $B' = B - \{k+1\}$ is also strict. Therefore, $SUM(B') = SUM(B) - s_{k+1}$ and $A_k(SUM(B')) \leq SP(B') \leq 1$ by induction hypothesis. Accordingly, $TMP_{k+1}(SUM(B)) \leq SUM(B) + \rho_{k+1} \cdot (SP(B') - SUM(B')) = SP(B)$. But $SP(B) < 1$ because B is a non-empty strict set. This implies $A_{k+1}(SUM(B)) = \min\{A_k(SUM(B)), TMP_{k+1}(SUM(B))\} \leq SP(B)$.

Q.E.D.

Theorem 2 *Let $A_n(r') = \min_{0 \leq r < 1} A_n(r)$. $C_n(r')$ is optimal and $SP(C_n(r')) = A_n(r')$.*

Proof:
It is clear $A_n(0) = 1$. Therefore, $A_n(r') < \infty$. Then from Lemma 1, $SP(C_n(r')) = A_n(r')$. From Theorem 1, there exists a strict optimal set B. From Lemma 2, $A_n(r') = \min_{\forall r} A_n(r) \leq A_n(SUM(B)) \leq SP(B)$, which implies $C_n(r')$ must be optimal.

Q.E.D.

4.3 IMPLEMENTATION AND EXAMPLE

P-SPO, the dynamic programming algorithm computing the recursive relations for $A_i(r)$ and $C_i(r)$, is listed in the following:

ALGORITHM P-SPO

```
0)   $1 \leq i \leq n$, input( $s_i$, $\rho_i$)
1)   $\forall r \in \{\frac{1}{2^K}, \frac{2}{2^K}, \cdots, \frac{2^K-1}{2^K}\}$, A(r) ← ∞, C(r) ← ⊥
2)   A(0) ← 1; C(0) ← ∅
3)   for i ← 1, to n
4)   for r ← $1 - \frac{1}{2^K}$, downto $s_i$, step - $\frac{1}{2^K}$
5)   begin
6)       if A(r - $s_i$) ≠ ∞, then
7)       begin
8)           TMP ← $r + \rho_i \cdot$ ( $A(r - s_i) - r + s_i$)
9)           if TMP < min{A(r), 1} then
                 A(r) ← TMP,
                 C(r) ← C($r - s_i$) ∪ {i}
10)      end{if}
11)  end{for}
12)  Find r such that A(r) is minimum.
13)  output (C(r), A(r))
```

In P-SPO, $A_i(r)$ and $C_i(r)$ are recorded in two arrays, namely, A(r) and C(r), $0 \leq r < 1$. The recursive relations are evaluated in the double loops starting on lines (3) and (4). During iteration $i = \iota$ and $r = \gamma$, $A_\iota(\gamma)$ and $C_\iota(\gamma)$ are computed and stored in $A(\gamma)$ and $C(\gamma)$. Note that the **"for"** loop in line (4) has to iterate in the **"downto"** direction for the following reason: In order to compute $TMP_\iota(\gamma)$, $A_{\iota-1}(\gamma-s_\iota)$ and $C_{\iota-1}(\gamma-s_\iota)$ have to be accessed. These values should be kept in $A(\gamma - s_\iota)$ and $C(\gamma-s_\iota)$, respectively. The **"downto"** direction guarantees that these two entries will not be over-written by $A_\iota(\gamma-s_\iota)$ and $C_\iota(\gamma-s_\iota)$ until a later iteration.

P-SPO takes $F = \{ (s_1, \rho_1), (s_2, \rho_2), \cdots, (s_n, \rho_n)\}$ as its input. It outputs an optimal set C(r) together with the corresponding minimum value of the objective function, A(r). In P-SPO we use a special symbol ⊥ to denote "undefined".

Example 1 *Let $|R_0|$, C_0, D_0 be 5, 2, and 1, respectively. There are four semi-joins, namely, $R_i \rightarrow R_0$,* *i = 1, 2, 3, 4, which can be used to reduce R_0. The transmission cost of them are 5, 2.5, 1.25, and 1.25, respectively. Their selectivities are 0.45, 0.6, 0.7, and 0.6.*

From the above data, we can construct F = { (1/2, 0.45), (1/4, 0.6), (1/8, 0.7), (1/8, 0.6) }. Notice that the precision is 3. We need to compute $A_i(r)$ for $1 \leq i \leq 3$ and r∈ {0, 1/8, 2/8, ⋯, 7/8}. Initially, $A_0(0)$=1 and $C_0(0) = \emptyset$. All other $A_0(r)$'s equal ∞ and all other $C_0(r)$'s equal ⊥. The computation of $A_i(r)$ and $C_i(r)$ is summarized in the following two tables:

$A_i(r) =$

$r \backslash i$	1	2	3	4
0	1	1	1	1
$\frac{1}{8}$	∞	∞	0.825	0.725
$\frac{2}{8}$	∞	0.85	0.85	0.67
$\frac{3}{8}$	∞	∞	0.795	0.735
$\frac{4}{8}$	0.95	0.95	0.95	0.752
$\frac{5}{8}$	∞	∞	0.94	0.895
$\frac{6}{8}$	∞	∞	∞	0.939
$\frac{7}{8}$	∞	∞	∞	∞

$C_i(r) =$

$r \backslash i$	1	2	3	4
0	∅	∅	∅	∅
$\frac{1}{8}$	⊥	⊥	{3}	{4}
$\frac{2}{8}$	⊥	{2}	{2}	{3, 4}
$\frac{3}{8}$	⊥	⊥	{2,3}	{2, 4}
$\frac{4}{8}$	{1}	{1}	{1}	{2, 3, 4}
$\frac{5}{8}$	⊥	⊥	{1, 3}	{1, 4}
$\frac{6}{8}$	⊥	⊥	⊥	{1, 3, 4}
$\frac{7}{8}$	⊥	⊥	⊥	⊥

We explain some entries in the above tables. First consider (i,r) = (2, 6/8). $\text{TMP}_2(6/8) = \frac{6}{8} + 0.6 \cdot (A_1(\frac{6}{8}-\frac{1}{4})-\frac{6}{8}+\frac{1}{4}) = 1.02 > 1$. Therefore, $A_2(\frac{6}{8}) = \infty$. Next consider (i, r) = (4, 3/8). $TMP_4(\frac{3}{8}) = \frac{3}{8} + 0.6 \cdot (A_3(\frac{1}{4}) - \frac{1}{4}) = 0.735 < A_3(\frac{3}{8})$. Thus $A_4(\frac{3}{8}) = 0.735$.

P-SPO outputs ({3, 4}, 0.67). The corresponding total transmission cost TS({3, 4}) is $C_0 \cdot \mid R_0 \mid \cdot 0.67 + D_0 = 7.7$.

□

4.4 THE COMPLEXITY OF P-SPO

We assume it takes constant time to perform a rational number addition, subtraction, or multiplication. It is also assumed that it takes constant memory space to store any rational number. The space complexity of P-SPO is dominated by the storage requirement of the array C(). The index r ∈ $\{0, \frac{1}{2^K}, \frac{2}{2^K}, \cdots, \frac{2^K-1}{2^K}\}$, where K is the precision. Each entry in C() requires n bits. Therefore, the total space complexity is $O(2^K \cdot n)$.

The time complexity is dominated by the execution of the double loop at line (3) and (4). The execution time for each iteration is constant. Therefore, the total time complexity is $O(2^K \cdot n)$.

5 ONE-SHOT SEMI-JOIN EXECUTION FOR QUERY OPTIMIZATION

We shall consider using one-shot semi-join execution for query optimization. In the following we first model the queries. Then the one-shot semi-join query optimization problem will be formulated into a mathematical programming problem. Finally, we provide a strategy to find the optimal solution.

5.1 THE MODEL OF QUERIES

A query, denoted (TL, Q), consists of two components: the target list (TL), and the qualification (Q). The target list contains target attributes that are of interest to the query, i.e., attributes that will appear in the answer. The qualification is a conjunction of clauses which describe the query. We assume that after initial local processing, all clauses in a qualification are of the form $R_i.a_j = R_k.a_h$.

The closure Q* of a qualification Q is defined as the qualification with the minimum number of clauses such that (1) all clauses in Q are in Q*, and (2) if $R_i.a_m = R_j.a_n$ and $R_j.a_n = R_k.a_o$ are in Q, then $R_i.a_m = R_k.a_o$ is in Q*.

The undirected join graph G(V,E) of a query (TL,Q) is defined as follows: V is the set of relations involved in the query. E is the set of edges such that (R_i, R_k) ∈ E if for some a_j and a_h, $R_i.a_j = R_k.a_h$ is in Q*.

Associated with each edge (R_i, R_j) ∈ E there are two semi-joins, namely, $R_i \rightarrow R_j$ and $R_j \rightarrow R_i$. As defined in section 3, u_i^j and ρ_i^j denote the transmission cost and selectivity of $R_i \rightarrow R_j$, respectively.

5.2 FORMULATION

Given a query graph G(V,E) with k edges, There are 2k semi-joins, two corresponding to each edge. The problem is to choose to execute a subset of these semi-joins such that the overall transmission cost is minimized.

Let XJ be the set of indices of these 2k semi-joins, i.e., XJ = { (i,j) | $R_i \rightarrow R_j$ is a semi-join }. Let BJ denote the subset of XJ such that { $R_i \rightarrow R_j$ |

(i,j) $\in BJ\}$ is the set of semi-joins chosen to be executed. The total transmission cost for these semi-joins is $\sum_{(i,j)\in BJ} u_i^j$. After the execution of these semi-joins, the size of R_j becomes $(\prod_{\forall i,(i,j)\in BJ} \rho_i^j)\cdot \mid R_j \mid$. Thus the transmission cost for sending R_j to the final processing site is $C_j \cdot (\prod_{\forall i,(i,j)\in BJ} \rho_i^j)\cdot \mid R_j \mid + D_j$. Suppose there are n relations, R_1, R_2 $\cdots$, R_n, in the query graph. The overall transmission cost, including the cost of executing all the chosen semi-joins and sending all the relations to the final processing site, is therefore

$$TSJ(BJ) = \sum_{(h,k)\in BJ} u_h^k + \sum_{j=1}^{n}(C_j \cdot \mid R_j \mid \cdot \prod_{\forall i,(i,j)\in BJ} \rho_i^j + D_j) \qquad (3)$$

To summarize the discussion above, the one-shot semi-join query optimization problem can be formulated into the following mathematical programming problem, JSPO.

Definition 3 *(JSPO)*
Given $FJ = \{\ (u_i^j, \rho_i^j) \mid (i,j) \in XJ\}$, *and* $M = \{\ (C_j, D_j, \mid R_j \mid) \mid 1 \le j \le n\}$, *find* $BJ \subset XJ$ *such that* *TSJ(BJ) is minimized.*

5.3 SOLVING JSPO

A Divide-and-Conquer algorithm based on SPO can be developed to solve JSPO.

TSJ(BJ) can be written as

$$TSJ(BJ) = \sum_{j=1}^{n}(C_j \cdot SP_j(BJ_j)\cdot \mid R_j \mid + D_j)$$

where

$$SP_j(BJ_j) = \sum_{(i,j)\in BJ_j} \frac{u_i^j}{C_j \cdot \mid R_j \mid} + \prod_{(i,j)\in BJ_j} \rho_i^j$$
$$BJ_j = \{(i,j) \mid \forall i, (i,j) \in BJ\}$$

Notice $\{BJ_j\}$ is a partition of BJ. TSJ(BJ) achieves minimum iff all $SP_j(BJ_j)$'s are minimized simultaneously. Conversely, if one can find X_1, X_2, $\cdots$, X_n, such that $SP_1(X_1)$, $SP_2(X_2)$, $\cdots$, $SP_n(X_n)$ are minimum, $\bigcup_{j=1}^{n} X_j$ is an optimal BJ. $SP_j()$ can be solved by using P-SPO if we let $s_i = \frac{u_i^j}{C_j \cdot |R_j|}$ and $\rho_i = \rho_i^j$. Therefore, JSPO can be solved by applying P-SPO to reduce each relation in the query.

6 THE TRUNCATION ERRORS

In practice, we may not always be able to find a small precision number K such that each parameter can be precisely represented. One alternative is to choose a reasonable K and to truncate all the parameters whose precision is greater than $\frac{1}{2^K}$. We can then use P-SPO as a heuristic to solve the problem.

Choosing K is crucial to the precision of the fixed-precision algorithm. In this section, we shall analyze the relationship between K and the errors caused by truncation.

We consider the single-relation one-shot semi-join execution first. P-SPO requires all s_i's be of fixed precision. There are no precision constraints on ρ_i's. The errors are therefore due to the truncation of s_i's only. Let F $= \{(s_1, \rho_1), (s_2, \rho_2), \cdots, (s_n, \rho_n)\}$ be the original problem and F$' = \{(s_1', \rho_1), (s_2', \rho_2), \cdots, (s_n', \rho_n)\}$ be the problem after the truncation. $s_i' \le s_i < s_i' + \frac{1}{2^K}$ because of the truncation. We use SP() and SP$'$() to represent the objective functions corresponding to the original problem and the problem after the truncation, respectively. Let B be an optimal set for the original problem and B$'$ be an optimal set derived by P-SPO for the problem after the truncation. Clearly SP(B) $\le$ SP(B$'$) and SP$'$(B$'$) $\le$ SP$'$(B). Since $s_i' \le s_i$ for all indices i, SP$'$(B) $\le$ SP(B). Combining these inequalities, we have

$$SP'(B') \le SP'(B) \le SP(B) \le SP(B') \qquad (4)$$

To compare the near-optimal set B$'$ (with respect to the original problem) and the optimal set B, we define the error of the truncation to be SP(B$'$) – SP(B). From (4), we have SP(B$'$) – SP(B) $\le$ SP(B$'$) – SP$'$(B$'$) $\le$ $\sum_{i\in B'}(s_i - s_i') \le \frac{n}{2^K}$, where K is the precision chosen for the truncation. In other words, if we use P-SPO as a heuristic, the strategy derived will incur at most $C_0 \cdot \mid R_0 \mid \frac{n}{2^K}$ units more transmission cost than the optimal strategy.

We now consider the one-shot semi-join query optimization problem. As in section 5, a join graph with n relations can be partitioned into n single-relation processing problems. Let n_i denote the number of candidate semi-joins for the ith single-relation subproblem. The overall truncation error is bounded by the summation of the truncation errors of all the sub-problems. Accordingly, the difference, in terms of transmission cost, between the strategy derived and the optimal strategy is no more than $\sum_{i=1}^{n} C_i \cdot \mid R_i \mid \frac{n_i}{2^K}$.

7 CONCLUSIONS AND FUTURE RESEARCH

We introduced the one-shot semi-join execution strategy for distributed query processing. This method allows parallelism and processes multiple semi-joins simultaneously. Specifically, the semi-join processing contains three phases, namely, projection, transmission, and reduction. During the projection phase, all semi-join projections from the same relations are generated at a time. These projections are then transmitted over the communication channel during the transmission phase. Finally, all semi-joins to the same relations are executed in the reduction phase.

Based on one-shot semi-join execution, we studied the distributed query optimization problem and tried to minimize the overall transmission cost. This problem has been shown to be NP-Hard. Nevertheless, we developed a polynomial-time algorithm, assuming the costs of semi-joins are of fixed precision. For situations where the fixed-precision constraint does not apply, we truncate the parameters and apply the algorithm as a heuristic to find near-optimal solutions. We analyzed the truncation errors and found that the solutions derived are close to the optimal one.

Some related problems are under investigation. We are using the fixed-precision constraint to solve other intractable query processing problems. Also, we are studying the global semi-join optimization problem. We propose one possible processing method, namely, hashing, in this paper. We intend to develop other methods and compare them under different scenarios.

References

[1] P.M.G. Apers, A.R. Hevner, and S.B. Yao. Optimization algorithms for distributed queries. *IEEE Trans. on Software Engineering*, SE-9(1):57–68, Jan. 1983.

[2] P.A. Bernstein and D.M. Chiu. Using semi-joins to solve relational queries. *JACM*, 28(1):25–40, Jan. 1981.

[3] P.A. Bernstein and N. Goodman. The power of natural semijoins. *SIAM J. Comput.*, 10(4):751–771, Nov. 1981.

[4] P.A. Bernstein et al. Query processing in a system for distributed databases (SDD-1). *ACM Trans. on Database Systems*, 6(4):602–625, Dec 1981.

[5] D. Brill, M. Templeton, and C.T. Yu. Distributed query processing strategies in Mermaid, a frontend to data management systems. In *Proc. IEEE Data Engineering Conf.*, Feburary 1984.

[6] S. Ceri and G. Pelagatti. *Distributed Databases: Principles and Systems.* McGraw-Hill, 1984.

[7] A.L.P. Chen, D. Brill, M. Templeton, and C.T. Yu. Distributed query processing in a multiple database system. *IEEE Journal on Selected Areas in Communications, special issue on Databases in Communications Systems*, April 1989.

[8] A.L.P. Chen and V.O.K. Li. Improvement algorithms for semi-join query processing programs in distributed database systems. *IEEE Trans. on Computers*, C-33(11):959–967, November 1984.

[9] D.M. Chiu, P.A. Bernstein, and Y.C. Ho. Optimizing chain queries in a distributed database system. *SIAM J. COMPUT.*, 13(1):116–134, February 1984.

[10] W. Perrizo and C. Chen. Composite semijoins in distributed query processing. *Information Sciences*, March 1990.

[11] S. Pramanik and D. Vineyard. Optimizing join queries in distributed databases. *IEEE Trans. on Software Engineering*, SE-14(9):1319–1326, September 1988.

[12] M. Templeton, D. Brill, A.L.P. Chen, S. Dao, and E. Lund. Mermaid— experiences with network operation. In *Proc. 2nd IEEE Data Engineering Conf.*, Feburary 1986.

[13] C. P. Wang. The complexity of processing tree queries in distributed databases. In *Proc. of the 2nd Symposium on Parallel and Distributed Processing*, 1990.

[14] E. Wong and K. Youssefi. Decomposition — a strategy for query processing. *ACM Trans. on Database Systems*, 1976.

[15] C.T. Yu, K. Guh, and A.L.P. Chen. An integrated algorithm for distributed query processing. In *Proc. IFIP Conference on Distributed Processing*, October 1987.

[16] C.T. Yu, Z. M. Ozsoyoglu, and K. Kam. Optimization of distributed tree queries. *J. Comput. Syst. Sci.*, 29(3):409–445, December 1984.

[17] C.T. Yu et al. Query processing in a fragmented relational distributed system: Mermaid. *IEEE Trans. on Software Engineering*, August 1985.