On the Complexity of Distributed Query Optimization

Chihping Wang, *Member, IEEE*, and Ming-Syan Chen, *Senior Member, IEEE*

Abstract—While a significant amount of research efforts has been reported on developing algorithms, based on joins and semijoins, to tackle distributed query processing, there is relatively little progress made toward exploring the complexity of the problems studied. As a result, proving NP-hardness of *or* devising polynomial-time algorithms for certain distributed query optimization problems has been elaborated upon by many researchers. However, due to its inherent difficulty, the complexity of the majority of problems on distributed query optimization remains unknown. In this paper we generally characterize the distributed query optimization problems and provide a frame work to explore their complexity. As it will be shown, most distributed query optimization problems can be transformed into an optimization problem comprising a set of binary decisions, termed Sum Product Optimization (SPO) problem. We first prove SPO is NP-hard in light of the NP-completeness of a well-known problem, Knapsack (KNAP). Then, using this result as a basis, we prove that five classes of distributed query optimization problems, which cover the majority of distributed query optimization problems previously studied in the literature, are NP-hard by polynomially reducing SPO to each of them. The detail for each problem transformation is derived. We not only prove the conjecture that many prior studies relied upon, but also provide a frame work for future related studies.

Index Terms—Distributed query optimization, semijoin processing, complexity, NP-hard problems, distributed databases.

1 Introduction

In a distributed relational database system, the processing of a query involves data transmission among different sites via a computer network. Since data are geographically distributed in such a system, the processing of a distributed query, as pointed out in [40], is composed of the following three phases:

1) *local processing phase* which involves all local processing such as selections and projections,
2) *reduction phase* where a sequence of reducers (i.e, semijoins and joins) is used to reduce the size of relations, and
3) *final processing phase* in which all resulting relations are sent to the assembly site where the final query processing is performed.

The objective taken in this context is mainly to reduce the communication cost required for data transmission [5], which is key to the performance of distributed database systems [2], [22], [36].

Clearly, a straightforward approach to processing a distributed query would be sending all relations directly to the assembly site, where all joins are performed. This naive method, however, is unfavorable due to its high transmission overhead and little parallelism exploitable. Significant research efforts have been focused on the problem of reducing the amount of data transmission required for phases 2) and 3) of distributed query processing [5], [14], [20], [27], [34], [39]. Specifically, two approaches have been employed, namely, semijoin [3] and join sequence [24].

The semijoin operation has received considerable attention and been extensively studied in the literature [2], [6], [25], [29], [31], [38]. The semijoin operation from R_i to R_j, denoted by $R_j \ltimes R_i$, is defined as follows: Project R_i on the join attribute of the join between R_i and R_j first, and then ship this projection to the site of R_j to remove nonmatching tuples from R_j. The transmission cost of sending R_j to the site of R_i for the join $R_i \bowtie R_j$ can thus be reduced. Illustrative examples of semijoin and join operations are given in Fig. 1, where the join attribute between R_1 and R_2 is attribute B. Semijoin based query optimization methods reduce the data transmission cost at the expense of increasing the processing overhead. How to identify and apply profitable semijoins to optimize distributed query processing has been explored in many prior studies [1], [3], [4], [22].

In addition to semijoins, join operations can also be used as reducers in processing distributed queries [10], [11], [26]. As shown in [10], [11], judiciously applying join operations as reducers can reduce the amount of data transmission required. Using join reducers, a query is translated into a sequence of joins, and each join is implemented locally by shipping one of the operand relations to the site of the other operand so as to exploit parallelism and minimize the processing overhead. Moreover, joins and semijoins can be combined to form an integrated scheme to further improve distributed query processing [9], [10], [26]. As pointed out in [10], the approach of combining join and semijoin operations as reducers can result in more beneficial semijoins due to the inclusion of join reducers. Also, this approach can reduce the communication cost further by taking advantage of the removability of pure join attributes.[1] An algorithm on

• C. Wang is with Informix Software, Inc., 4100 Bohannon Drive, Building 4600/2, Menlo Park, CA 94025. E-mail: cpwang@informix.com.
• M.-S. Chen is with the Electrical Engineering Department, National Taiwan University, Taipei, Taiwan, ROC. E-mail: mschen@cc.ee.ntu.edu.tw.

Manuscript received Nov. 8, 1994.
For information on obtaining reprints of this article, please send e-mail to: transkde@computer.org, and reference IEEECS Log Number K96055.

1. Pure join attributes are those which are used in join predicates bu not part of the output attributes.

R_1

A	B
a_1	b_1
a_2	b_1
a_2	b_2
a_2	b_4
a_3	b_4
a_4	b_7
a_4	b_9

R_2

B	C
b_1	c_2
b_2	c_1
b_2	c_4
b_5	c_2
b_6	c_4
b_7	c_2
b_8	c_3

(a) Relations R_1 and R_2.

B	C
b_1	c_2
b_2	c_1
b_2	c_4
b_7	c_2

(b) R_2 after semijoin $R_j \ltimes R_i$ on attribute B.

A	B	C
a_1	b_1	c_2
a_2	b_1	c_2
a_2	b_2	c_1
a_2	b_2	c_4
a_4	b_7	c_2

(c) Resulting relation from $R_i \bowtie R_j$.

Fig. 1. Examples of semijoin and join operations.

interleaving a join sequence with semijoins to minimize the amount of data transmission in distributed query processing was developed in [9].

While a significant amount of research efforts has been reported on developing algorithms, based on joins and semijoins, to tackle distributed query processing, there is relatively little progress made toward exploring the complexity of, as well as characterizing, the problems studied. As a matter of fact, up to today, only the complexities of very few distributed query optimization problems have been explicitly understood; either proved to be NP-hard or shown polynomially solvable. Although it has been a conjecture that several problems on distributed query optimization are NP-hard, due to its inherent difficulty, there is no general proof for this conjecture thus far. Consequently, while being aware of the difficulty of these problems, researchers in this field have mostly worked on either proving NP-hardness of *or* developing polynomial-time algorithms for certain distributed query optimization problems. It has been shown that optimization for cyclic queries is NP-hard [20]. Also, there have been results reported in polynomially optimizing semijoin sequences to process certain tree queries, such as star and chain queries [7], [12]. Subject to some specific constraints, it was shown that finding the optimal distributed join sequence is NP-hard [24]. In addition, it has been proved that problems on optimizing some special types of distributed queries, including queries involving set operations [17], those with joins on fragmented or hash-partitioned relations [8], [32], and semantic queries (in terms of eliminating unnecessary joins) [33], are NP-hard. Note, however, that these results, while applicable to certain cases, were individually derived through different techniques, and thus do not in general provide a frame work to characterize the complexity of distributed query optimization. As a result, the complexity of the majority of problems on distributed query optimization remains unknown. The necessity of resorting to heuristics [2], [11], [28], [31], [38], [40] and exponential-time algorithms [12], [13], [29] to deal with these optimization problems is thus left unjustified.

To remedy this, we shall in this paper generally characterize the distributed query optimization problems and provide a frame work to explore their complexity. As it will be shown later, most distributed query optimization problems can be transformed into an optimization problem comprising a set of binary decisions. Those decisions determine two costs: one in a summation form and the other in a product form. This optimization problem is therefore termed Sum Product Optimization (SPO) problem. We first prove SPO is NP-hard by showing its decision version (i.e., problem with a *yes* or *no* answer), denoted by SPD, is NP-complete. Specifically, we prove that a well-known NP-complete problem, Knapsack (KNAP), can be polynomially reduced to SPD, thus proving that SPD is NP-complete. The NP-hardness of SPO hence follows [16]. It is noted that KNAP is only NP-complete in the weak sense. That is, KNAP can be solved in pseudo-polynomial time and is able to be approximated with a high degree of accuracy. This fact explains the reason that many high quality heuristics have been developed in the literature for distributed query processing.

Problems SPO, SPD, and KNAP will be formally defined in Section 3.1. Then, using the NP-hardness of SPO as a basis, we prove that the following five classes of distributed query optimization problems, which cover the majority of distributed query optimization problems previously studied in the literature, are NP-hard by polynomially reducing SPO to each of them, thereby justifying the necessity of using heuristics to solve these problems. The detail for each problem transformation will be derived in Section 4. The five classes of distributed query optimization problems are given below.

Five classes of distributed query optimization problems:

1) **Local optimization of semijoins**: Determine the optimal set of semijoins to reduce a single relation (studied in [3], [31], [37]).
2) **Join sequence optimization**: Determine the optimal join sequence to transfer relations to the assembly site (studied in [11], [24]).
3) **Relation semijoins on broadcasting networks**: Solve the semijoin optimization problem on broadcasting networks (studied in [21], [28], [30]).
4) **Single-reducer tree queries**: Determine the minimal-cost semijoin sequence to fully reduce the root relation of a tree query (studied in [7], [13], [40]).
5) **Full-reducer tree queries**: Determine the minimal-cost semijoin sequence to fully reduce all relations in a tree query (studied in [13], [19], [29], [40]).

The contributions of this paper are twofold. We not only prove a general optimization problem, SPO, is NP-hard, but also in light of this result, show that most problems on distributed query optimization are NP-hard for their reducibility from SPO via polynomial transformation, thus proving the conjecture that many prior studies relied upon. It is worth mentioning that the usefulness of NP-hardness of SPO proved in this paper is not confined to the context of distributed query optimization, and is in fact applicable to solving the complexity of many other optimization problems. To the best of our knowledge, despite of its importance, there is no prior work on generally characterizing the distributed query optimization problems, let alone providing a frame work to explicitly prove their complexity. This feature distinguishes this paper from others.

This paper is organized as follows. Notation, assumptions, and cost models are given in Section 2. In Section 3, we first describe problems SPO, SPD, and KNAP, and then prove SPD is NP-complete by polynomially reducing KNAP to SPD. SPO is thus proved NP-hard. In Section 4, we use the NP-hardness of SPO as a basis to prove that five classes of problems on distributed query optimization are NP-hard. This paper concludes with Section 5. For better readability, a list of symbols is given in the Appendix.

2 Preliminary

2.1 Notation and Assumptions

A query, denoted by (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 to appear in the answer. To facilitate our presentation, it is assumed that a query is in the form of conjunctions of equi-join predicates and all attributes are renamed in such a way that two join attributes have the same attribute name if and only if they have a join predicate between them. Specifically, the closure Q^* of a qualification Q is defined as the qualification with the minimal number of clauses such that

1) all clauses in Q are in Q^*, and
2) if $a = b$ and $b = c$ are in Q, then $a = c$ is in Q^*.

When multiple copies of a relation exist, we assume that one copy has been preselected.

A join query graph can be denoted by a graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. Each node in a query graph represents a relation. Two nodes R_i and R_j are connected by an edge, denoted by $(R_i, R_j) \in E$, if there exists a join predicate $R_i.a_j = R_j.a_j$ on some attribute a_j of the two relations. $R_i \bowtie R_j$ denotes the join between R_i and R_j, and $R_i \ltimes R_j$ means that R_j applies a semijoin to reduce the cardinality of R_i. $|R_i|$ is used to denote the cardinality of relation R_i.

2.2 Cost Model

As in most previous work in distributed databases [5], [39], we assume that the cost of executing a query can mainly be expressed in terms of the total amount of inter-site data transmission required. Note that the objective on merely minimizing the total amount of intersite data transmission assumed in this paper should not be viewed as a deficiency of this study. Rather, following directly from our results on the primitive model, it can be proved by restriction [16] that the corresponding optimization problems on more complicated models, such as those on including the computational cost and on minimizing the response time, are NP-Complete, showing the general applicability of this study. The cost of sending R_i to the site of R_j is expressed as $CJ_i^j|R_i|$, where CJ_i^j is the transmission cost per tuple of sending R_i to R_j. Notice that for simplicity we do not consider the transmission setup cost, which is a constant term. It can be seen that our results in this paper remain valid in the presence of such a setup cost. We use CJ_i^A to denote the transmission cost per tuple of sending R_i to the assembly site. Also, $R_i.a_{t(j)}$ is used to denote the projection of R_i on the join attribute $a_{t(j)}$ for semijoin $R_i \ltimes R_j$. Similarly, the cost of sending $R_i.a_{t(j)}$ to R_j is expressed as $R_i \bowtie R_j$, where CS_i^j is the transmission cost per tuple of sending $R_i.a_{t(j)}$ to R_j.

The selectivity model for join and semijoin operations employed in this paper to assess their cost and effect is the same as those in prior studies [5], [39]. Under this model, it is assumed that there is a number σ_i^j, called join selectivity, associated with each join $R_i \bowtie R_j$ such that $|R_i \bowtie R_j| = \sigma_i^j \cdot |R_i| \cdot |R_j|$. Note that $\sigma_i^j = \sigma_j^i$. On the other hand, the semijoin selectivity, ρ_i^j, is associated with each semijoin $R_i \ltimes R_j$, such that $|R_i \ltimes R_j| = \rho_j^i \cdot |R_i|$. Both σ_i^j and ρ_i^j are rational numbers between 0 and 1. Note that several formulae to determine the selectivities in the presence of various data skew have been derived in [15], whose discussion is beyond the scope of this paper. As can be seen later, our derivation on complexity does not assume any particular data distribution, and is in fact orthogonal to the method used to determine the selectivities. Same as in prior work on distributed query processing, we assume that the data transmission cost is additive, i.e., the total transmission cost is determined by the sum of all transmission costs of joins and semijoins.

3 Sum Product Optimization Problem

We shall describe in Section 3.1 the Sum Product Optimization (SPO) problem, which most distributed query optimization problems can be transformed into. Then, we prove in Section 3.2 that SPD, the decision version of SPO, can be polynomially reduced from a well-known NP-complete problem, KNAP, thus proving the NP-completeness of SPD. The NP-hardness of SPO follows.

3.1 Problem Formulation

SPO is a problem consisting of n binary decisions, denoted by $\{d_1, d_2, ..., d_n\}$ where each $d_i \in 0, 1$. Associated with the ith decision d_i, there is a pair of parameters (s_i, p_i), representing the cost and the profit, respectively, of d_i. s_i and p_i

are rational numbers between zero and one. If all d_i s are 0, then the default cost is one. To minimize this cost, some decisions are set to be 1s in such a way that if $d_i = 1$, the default cost is reduced by a factor of p_i at the expense of incurring an extra cost s_i. The objective is to find the optimal decisions such that the sum of the reduced default cost and the extra costs incurred is minimized. Formally, SPO is defined as follows.

DEFINITION 1 (Problem SPO). *Given* $F = \{(s_1, p_1), (s_2, p_2), \ldots, (s_n, p_n)\}$, *where* $s_i, p_i \in Q^{(0,1]}$ *(i.e., rational in* (0, 1]), *determine* $B \subset \{1, 2, \ldots, n\}$ *such that*

$$SP(B) = \sum_{i \in B} s_i + w \prod_{i \in B} p_i$$

is minimized, where w *is a weighting factor between costs and profits.*

In what follows, we shall use *PROD(B)* to denote $\prod_{i \in B} p_i$ and *SUM(B)* to denote $\sum_{i \in B} s_i$. In order to prove that the optimization problem SPO is NP-hard, we shall first introduce its corresponding decision problem (i.e., problem with a *yes* or *no* answer), denoted by SPD, and then prove that the decision problem SPD is NP-complete. Without loss of generality, assuming $w = 1$, SPD can then be defined as follows.

DEFINITION 2 (Problem SPD). *Given a rational number* c *and* $F = \{(s_1, p_1), (s_2, p_2), \ldots, (s_n, p_n)\}$, *where* $s_i, p_i \in Q^{(0,1]}$, *is there* $B \subset \{1, 2, \ldots, n\}$ *such that* $SP(B) < c$?

Clearly, if there is a polynomial-time algorithm for SPO, SPD can be answered in polynomial time. Equivalently, if SPD cannot be answered in polynomial time, then there is no polynomial-time algorithm for SPO. This means that if SPD is NP-complete, then SPO is NP-hard. In view of this, we shall show that SPD can be polynomially reduced from a well-known NP-complete problem, Knapsack (KNAP), thus proving that SPD is NP-complete [16]. SPO is therefore NP-hard. The definition of KNAP is given below [23].

DEFINITION 3 (Problem KNAP). *Given an integer* k, *and a set* $K = \{a_1, a_2, \ldots, a_n\}$, *where* a_i *s are positive integers and* $a_i \le k$, *for* $1 \le i \le n$, *is there a subset* $C \subset \{1, 2, \ldots, n\}$ *such that*

$$k = \sum_{i \in C} a_i?$$

A polynomial reduction from KNAP to SPD is derived as follows.

DEFINITION 4 (Polynomial reduction from KNAP to SPD). *Given an instance of* KNAP *with* k *and* K, *we construct an instance of* SPD *according to the following steps:*

1) *Calculate* $\epsilon = \frac{1}{72k^3}$.
2) *Let* $e(x, n)$ *denote the* nth *degree Taylor's polynomial of the natural exponential function at* x, *and* $len(y)$ *denote the length of the binary representation of integer* y. *Construct* $F = \{(s_1, p_1), (s_2, p_2), \ldots, (s_n, p_n)\}$, *where*

$$s_i = \frac{a_i}{k} e(-1,\ 6 + len(n) + 3 \cdot len(k)),$$

$$p_i = e\left(-\frac{a_i}{k},\ 6 + 2 \cdot len(n) + 3 \cdot len(k)\right).$$

3) *Let* $c = 2 \cdot e(-1, 6 + 3 \cdot len(k)) + 2 \cdot \varepsilon$.

The above polynomial reduction can be explained as follows. Consider the function $f(x) = x + e^{-e \cdot x}$. $f(x)$ reaches its minimum $\frac{2}{e}$ if and only if $x = \frac{1}{e}$. Given an instance of KNAP, if we choose $s_i = \frac{a_i}{k \cdot e}$, $p_i = e^{-\frac{a_i}{k}}$, and $c = \frac{2}{e}$, then

$$SP(B) = f\left(\frac{\sum_{j \in B} a_j}{k \cdot e}\right).$$

Thus, $SP(B)$ reaches its minimum $\frac{2}{e}$ if and only if $\sum_{j \in B} a_j = k$. Note that s_i, p_i, and c are not rational numbers as required by Definition 2. Hence, we use Taylor's polynomial to approximate $\frac{a_i}{k \cdot e}$, $e^{-\frac{a_i}{k}}$, and $\frac{2}{e}$. By analyzing the errors of this approximation, it will be shown that the reduction in Definition 4 is a polynomial reduction from KNAP to SPD, thus proving SPD is NP-complete. The NP-hardness of SPO follows.

3.2 Proof of NP-hardness of SPO

In the following discussion, s_i, p_i, ε, and c are parameters as defined in Definition 4. Let $F(B)$ be

$$\sum_{i \in B} \frac{a_i}{e \cdot k} + \prod_{i \in B} e^{-\frac{a_i}{k}}.$$

The approximation error of SP(B) is analyzed first.

LEMMA 1. $\forall B \subset \{1, 2, \cdots, n\}$, $|SP(B) - F(B)| < \epsilon$.

PROOF. First, consider the error of SUM(B). It follows from Taylor's formula that

$$\left|s_i - \frac{a_i}{e \cdot k}\right| \le \frac{a_i}{k}\left(\frac{1}{(6 + len(n) + 3 \cdot len(k))!}\right) < \frac{1}{144 \cdot n \cdot k^3}.$$

Accordingly,

$$\left|\sum_{i \in B} s_i - \sum_{i \in B} \frac{a_i}{e \cdot k}\right| < \frac{1}{144 \cdot k^3}. \tag{1}$$

Next, consider the error of PROD(B).

$$\left|p_i - e^{-\frac{a_i}{k}}\right| \le \frac{1}{(6 + 2 \cdot len(n) + 3 \cdot len(k))!} \le \frac{1}{720 \cdot n^2 \cdot k^3},$$

which implies

$$\left|\frac{p_i}{e^{-\frac{a_i}{k}}} - 1\right| < \frac{e^{\frac{a_i}{k}}}{720 \cdot n^2 \cdot k^3}.$$

However, $e^{\frac{a_i}{k}} < e^1 < 3$. Therefore,

$$\left|\frac{p_i}{e^{-\frac{a_i}{k}}} - 1\right| < \frac{1}{144 \cdot n^2 \cdot k^3}$$

and

$$\left(1 - \frac{1}{144 \cdot n^2 \cdot k^3}\right)^n < \prod_{i \in B} \frac{p_i}{e^{-\frac{a_i}{k}}} < \left(1 + \frac{1}{144 \cdot n^2 \cdot k^3}\right)^n.$$

From

$$\left(1-\frac{1}{144\cdot n^2\cdot k^3}\right)^n = 1+\sum_{i=1}^{n}(-1)^i\cdot\binom{n}{i}\cdot\left(\frac{1}{144\cdot n^2\cdot k^3}\right)^i$$

$$> 1-\sum_{i=1}^{n}\frac{1}{144\cdot n\cdot k^3} = 1-\frac{1}{144\cdot k^3}$$

and

$$\left(1+\frac{1}{144\cdot n^2\cdot k^3}\right)^n = 1+\sum_{i=1}^{n}\binom{n}{i}\cdot\left(\frac{1}{144\cdot n^2\cdot k^3}\right)^i$$

$$< 1+\sum_{i=1}^{n}\frac{1}{144\cdot n\cdot k^3} = 1+\frac{1}{144\cdot k^3},$$

we have

$$\left|\left(\prod_{i\in B}\frac{p_i}{e^{-\frac{a_i}{k}}}\right)-1\right| < \frac{1}{144\cdot k^3}.$$

Note that

$$\prod_{i\in B} e^{-\frac{a_i}{k}} < 1.$$

Consequently,

$$\left|\prod_{i\in B}p_i - \prod_{i\in B}e^{-\frac{a_i}{k}}\right| < \frac{1}{144\cdot k^3}. \qquad (2)$$

Combining (1) and (2), we have

$$|SP(B)-F(B)| < \frac{2}{144\cdot k^3} = \epsilon,$$

thus proving this lemma. □

Next, the approximation error of $SP(B) - c$ is analyzed in the following lemma.

LEMMA 2.

$$\forall B\subset\{1, 2, \cdots, n\},\ F(B)-\frac{2}{e}-4\cdot\epsilon < SP(B)-c < F(B)-\frac{2}{e}.$$

PROOF. From Taylor's formula, it can be obtained that

$$\left|c-\left(\frac{2}{e}+2\cdot\epsilon\right)\right| < \frac{2}{(6+3len(k))!} < \epsilon.$$

From this inequality and Lemma 1, we have

$$(F(B)-\epsilon)-\left(\left(\frac{2}{e}+2\cdot\epsilon\right)+\epsilon\right) < SP(B)-c$$

$$< (F(B)+\epsilon)-\left(\left(\frac{2}{e}+2\cdot\epsilon\right)-\epsilon\right),$$

leading to this lemma. □

The following lemma specifies a lower bound of $F(B)$ when $\sum_{i\in B}a_i \neq k$.

LEMMA 3. *$\forall B\subset\{1, 2, \cdots, n\}$ if $\sum_{i\in B}a_i\neq k$, then $F(B) > \frac{2}{e}+4\cdot\epsilon$.*

PROOF. Define $f(x) = x + e^{-e\cdot x}$. Notice that

$$F(B) = f\left(\frac{\sum_{i\in B}a_i}{e\cdot k}\right).$$

$f(x)$ has the minimal value $\frac{2}{e}$, which only occurs at $x = \frac{1}{e}$. Also notice that $f(x)$ is concave upward and $f(x) \geq \min\{f(\frac{1}{e}+d), f(\frac{1}{e}-d)\}$ if $|x-\frac{1}{e}|\geq d$. Since a_is are integers,

$$\sum_{i\in B}a_i \neq k$$

implies

$$\left|\frac{\sum_{i\in B}a_i}{e\cdot k}-\frac{1}{e}\right| \geq \frac{1}{e\cdot k}.$$

Accordingly, $F(B) \geq \min\{f(\frac{1}{e}+\frac{1}{e\cdot k}), f(\frac{1}{e}-\frac{1}{e\cdot k})\}$.

It suffices to show that $f(\frac{1}{e}+\frac{1}{e\cdot k}) > \frac{2}{e}+4\cdot\epsilon$ and $f(\frac{1}{e}-\frac{1}{e\cdot k}) > \frac{2}{e}+4\cdot\epsilon$. $f(\frac{1}{e}+\frac{1}{e\cdot k}) = \frac{1}{e}\cdot(1+\frac{1}{k}+e^{-\frac{1}{k}})$. Using Taylor's polynomial to evaluate $e^{-\frac{1}{k}}$, we have $\frac{1}{k}+e^{-\frac{1}{k}} > 1+\frac{1}{6\cdot k^3}$. Consequently,

$$f\left(\frac{1}{e}+\frac{1}{e\cdot k}\right) > \frac{1}{e}\cdot\left(2+\frac{1}{6\cdot k^3}\right) > \frac{2}{e}+4\cdot\epsilon.$$

Similarly, it can be shown that $f(\frac{1}{e}-\frac{1}{e\cdot k}) > \frac{2}{e}+4\cdot\epsilon$. This lemma follows. □

From the above three lemmas, we have the following important theorem.

THEOREM 1. *Problem SPD is NP-complete.*

PROOF. Clearly, SPD is in NP because one can guess the set B first, then verify whether $SP(B)$ is less than c or not. We shall prove that the reduction in Definition 4 is a polynomial reduction from KNAP to SPD. The NP-completeness of SPD then follows from the fact that KNAP is NP-complete. Note that the reduction in Definition 4 has the property that $\forall i$, $0 < s_i, p_i < 1$. Therefore, (c, F) is an instance of SPD. It is clear that ε can be calculated in polynomial time. s_i can be calculated in polynomial time since there are only $6 + len(n) + 3\cdot len(k)$ terms in the Taylor's polynomial and each term can be calculated in polynomial time. For a same reason, p_i s and c can also be calculated in polynomial time, meaning that the reduction can be done in polynomial time.

To prove that the reduction effectively reduces KNAP to SPD, it suffices to show that $\forall C\subset\{1, 2, \cdots, n\}, \sum_{i\in C}a_i = k$ if and only if $SP(C) < c$. If $\sum_{i\in C}a_i = k$, then $F(C) = \frac{2}{e}$ and $SP(C) - c < 0$ from Lemma 2. On the other hand, if $\sum_{i\in C}a_i \neq k$, then from Lemmas 2 and 3, $0 < F(C) - \frac{2}{e} - 4\cdot\epsilon < SP(C) - c$, thus proving this theorem. □

Theorem 1 leads to the following corollary.

COROLLARY 1.1. *Problem SPO is NP-hard.*

4 Complexity of Distributed Query Optimization

In this section, we use the NP-hardness of SPO as a basis to prove that the following 5 classes of distributed query optimization problems are NP-hard:

1) local optimization of semijoins,
2) join sequence optimization,
3) relation semijoins on broadcasting networks,
4) single-reducer tree queries, and
5) full-reducer tree queries, by polynomially reducing SPO to each of them.

For each class of problems, we shall describe the corresponding problem reduction first, and then formally prove that the reduction derived is valid and can be done in polynomial time.

4.1 Local Optimization of Semijoins (Problem LO)

The query optimizer of SDD-1 [37] is among the first to apply semijoins to distributed query processing. The optimizer evaluates the benefit and cost of all candidate semijoins, performs the most profitable one, and updates the cardinality of the relation reduced by this semijoin accordingly. This procedure repeats until there is no profitable semijoin available. This approach is greedy and suboptimal.

A general version of the above approach is that the query optimizer computes for each relation R_i, instead of the most profitable semijoin, the most profitable set of semijoins to reduce R_i. Since all semijoins to R_i are considered at the same time, local optimality (with respect to R_i) can be attained. The solution obtained is local optimal in that only the reduction of one relation is taken into consideration at a time. Such a problem is hence called local optimization of semijoins, denoted by LO, and can be formally stated as follows.

Definition 5 (Problem LO) [31]. *Given R_0, the relation to be reduced, and n candidate semijoins:*

$$R_0 \ltimes R_1, R_0 \ltimes R_2, \ldots, R_0 \ltimes R_n,$$

determine a subset of these semijoins such that the total transmission cost, including the cost of applying semijoins and that of shipping R_0 to the assembly site, is minimized.

From the selectivity and transmission cost model described in Section 2, the cost of semijoin $R_0 \ltimes R_i$ is formulated as $CS_i^0 \cdot |R_i.a_{t(0)}|$. Let $B = \{i \mid R_0 \ltimes R_i \text{ is chosen}\}$. After all the semijoins chosen for R_0 are applied, the cardinality of R_0 becomes $\prod_{i \in B} \rho_i^0 \cdot |R_0|$, where $|R_0|$ is the original cardinality of R_0. The transmission cost of sending R_0 to the assembly site is thus $CJ_0^A \cdot \prod_{i \in B} \rho_i^0 \cdot |R_0|$. The goal is to determine an optimal B to minimize:

$$\sum_{i \in B} \left(CS_i^0 \cdot |R_i.a_{t(0)}| \right) + CJ_0^A \cdot \prod_{i \in B} \rho_i^0 \cdot |R_0|.$$

Note that although this problem was studied in [31] and a heuristic was developed, its complexity was unanswered. Consequently, we derive the polynomial reduction from SPO to LO as described below.

<u>Description of reduction from SPO to LO:</u>

Problem setup: Given an instance of SPO, let $|R_0| = CJ_0^A = 1$. For $1 \le i \le n$, let $CS_i^0 = 1$, $|R_i.a_{t(0)}| = s_i$, and $\rho_i^0 = p_i$.

Decision: The ith decision, d_i, is set to 1 if $R_0 \ltimes R_i$ is selected.

Default cost: If all d_i s are 0, i.e., no semijoins are applied, then the cost of shipping R_0 to the assembly site is 1.

Extra cost: If d_i is 1, the extra cost will be the transmission cost of $R_0 \ltimes R_i$, i.e., s_i.

Difficulty: Applying a semijoin to R_0 will reduce the cost of shipping R_0 to the assembly site. However, the semijoin itself incurs an extra transmission cost.

Clearly, the above reduction from SPO to LO can be done in polynomial time. In light of this and the fact that SPO is NP-hard, we have the following lemma whose proof is straightforward, and thus omitted.

Lemma 4. *Problem LO is NP-hard.*

4.2 Join Sequence Optimization (Problem JSO)

Using the join sequence method for distributed query processing, some joins are performed locally first, and the resulting relations are then sent to the assembly site. In [24], join sequence optimization is formulated as a graph problem. Suppose there are n relations in the query. Construct a directed graph with $n + 1$ nodes, where there is one node corresponding to the assembly site A, and each of the remaining n nodes is one-to-one associated with a relation. An edge (R_i, R_j) means R_i can be sent to R_j to perform a join. An edge (R_i, A) means R_i can be sent to the assembly site directly. The objective is to find an inversely directed spanning tree toward A with the minimal transmission cost. Relations are transmitted to A along the path determined by the spanning tree. Each intermediate node on the path represents a join. An example spanning tree is shown in Fig. 2. Consider edge (R_3, R_4) for example. After R_1 and R_2 are joined with R_3, the cardinality of R_3 becomes $(\sigma_1^3 \cdot |R_1| \cdot |R_3|) \cdot |R_2| \cdot \sigma_2^3$.The transmission cost of (R_3, R_4) is therefore $(\sigma_1^3 \cdot |R_1| \cdot |R_3|) \cdot |R_2| \cdot \sigma_2^3 \cdot CJ_3^4$. The total cost of the spanning tree in Fig. 2 is thus equal to,

$$|R_1| CJ_1^3 + |R_2| CJ_2^3 + \left(|R_1| |R_2| R_3| \, \sigma_1^3 \sigma_2^3 \right) CJ_3^4$$
$$+ \left(|R_1| |R_2| |R_3| \sigma_1^3 \sigma_2^3 |R_4| \sigma_3^4 \right) CJ_4^A$$

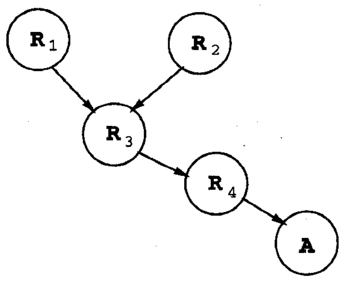


Fig. 2. An example spanning tree for join sequence optimization.

The join sequence optimization problem can now be formally defined as follows.

DEFINITION 6 (Problem JSO) [24]. *Given the following input parameters,*

1) $G = (V, E)$, *where G is a complete directed graph and* $V = \{R_1, R_2, \ldots, R_n, A\}$,
2) $|R_i|$, *the cardinality of* R_i, $1 \leq i \leq n$,
3) CJ_i^j, *the unit communication cost of edge*(R_i, R_j),
4) CJ_i^A, *the unit communication cost of edge*(R_i, A),
5) σ_i^j, *the join selectivity from* R_i *to* R_j,

the objective of JSO is to find an inversely directed spanning tree toward node A with the minimal transmission cost.

Note that although with some constraints on the solution space, certain versions of JSO can be proved to be NP-hard [24], those results do not in general prove that JSO itself is an NP-hard problem. We shall outline the mapping from SPO to JSO first, and then present a formal problem reduction.

Description of reduction from SPO to JSO:

Problem setup: Consider the directed graph in Fig. 3.

Decision: The *i*th decision, d_i, is set to 1 if R_i is sent to the assembly site directly. Otherwise, d_i is 0 and R_i is first joined with R_0.

Default cost: If all d_i s are 0, then all relations R_i, $1 \leq i \leq n$, are first joined with R_0. The result is then sent to the assembly site. The problem is so constructed that the communication cost from R_i to R_0 is negligible,[2] and the default cost is thus proportional to the cardinality of the join result. Without loss of generality, assume the default cost is one.

Extra cost: If d_i is 1, i.e., R_i is sent to the assembly site directly, the transmission cost is s_i. Since R_i is not first joined with R_0 anymore, the default cost is reduced by a factor of

$$\frac{1}{\sigma_i^0 \cdot |R_i|},$$

which is set to be p_i in our construction.

Difficulty: Sending relations directly to the assembly site is more expensive than sending them to R_0. However, joining relations with R_0 might increase the cardinality of the resulting relation, and also the communication cost required from the site of R_0 to A.

The formal problem reduction from SPO to JSO is given below.

DEFINITION 7 (Polynomial reduction from SPO to JSO). *Given an instance of SPO, we construct a directed graph as in* Fig. 3. *The parameters are defined as follows:*

1) $|R_i| = \frac{1}{p_i}$, $CJ_i^A = s_i \cdot p_i$, $\sigma_i^0 = 1$, $1 \leq i \leq n$.
2) $|R_0| = 1$, $CJ_0^A = \prod_{i=1}^n p_i$
3) *Let integers* $\nu(x)$ *and* $\delta(x)$ *denote, respectively, the numerator and denominator of the parameter x. In other words, x is represented by* $\frac{\nu(x)}{\delta(x)}$ *in the given instance of SPO. Define*

$$\Delta = \frac{1}{\delta(c) \cdot \prod_{i=1}^n \delta(s_i) \cdot \prod_{i=1}^n \delta(p_i)}.$$

Let $CJ_i^0 = p_i \cdot \frac{1}{(n+1)} \cdot \Delta$.

2. Such a setting is for ease of exposition. Nevertheless, it simplifies the problem studied. Note that is a simplified problem is NP-hard, its more complicated version can be proved to be NP-hard by restriction [16].

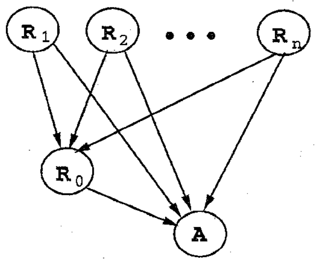


Fig. 3. A polynomial problem reduction from SPO to JSO.

We then have the following lemma which proves JSO is NP-hard.

LEMMA 5. *Problem JSO is NP-hard.*

PROOF. We shall prove that Definition 7 is a polynomial reduction from SPO to JSO, and this lemma follows from the fact that SPO is NP-hard. It is clear that the reduction in Definition 7 can be done in polynomial time. We then prove this reduction transforms SPO to JSO. Given $B = \{i \mid R_i \text{ is sent to A directly.}\}$, the total cost $T_{JS}(B)$ can be expressed as:

$$T_{JS}(B) = \sum_{i \in B} \left(CJ_i^A \cdot |R_i| \right) + \sum_{i \notin B} \left(CJ_i^0 \cdot |R_i| \right) + CJ_0^A \cdot |R_0| \prod_{i \notin B} \left(\sigma_i^0 \cdot |R_i| \right)$$
$$= SP(B) + \frac{1}{(n+1)} \cdot \Delta \cdot \left(\sum_{i \notin B} 1 \right)$$
$$< SP(B) + \Delta. \quad (3)$$

Next, it suffices to prove that

$$\exists B, SP(B) < c \Leftrightarrow \exists B', T_{JS}(B') < c.$$

Note that $T_{JS}(B) > SP(B)$. Therefore, $\forall B, SP(B) \geq c \Rightarrow \forall B, T_{JS}(B) \geq c$.

To prove the other direction, suppose there is B such that $SP(B) < c$. We claim $SP(B) \leq c - \Delta$. Then it can be obtained from(3) that $T_{JS}(B) < c$. Note that both $\frac{1}{\Delta} \cdot SP(B)$ and $\frac{1}{\Delta} \cdot c$ are integers because $\frac{1}{\Delta}$ eliminates the denominators of s_i, p_i, and c. Therefore, $SP(B) < c$ implies

$$\frac{1}{\Delta} \cdot SP(B) \leq \frac{1}{\Delta} \cdot c - 1.$$

We get $SP(B) \leq c - \Delta$, and this lemma follows. □

4.3 Relation Semijoin on Broadcasting Networks (Problem RSO)

As shown in [21], [28], [30], taking advantage of the broadcasting property of local area networks, the relation semijoin method can be applied to improve the query processing. Using this method, a distributed semijoin is accomplished by sending the entire relation, rather than the join attribute, to its recipient. The advantage is that multiple stations may receive the relation and apply different semijoins at the same time. Further, the assembly site also receives the relation broadcast, and each relation thus needs to be scanned only once. To optimize the transmission cost, the query optimizer has to determine the broadcasting order of the relations

Use CJ_i^A to denote the unit broadcasting cost of R_i. Let $q()$ be a permutation function on the numbers in $\{1, 2 \ldots, n\}$, which determines the broadcasting order of the relations. That is, $q(i) = j$ means R_j is the ith relation to be broadcast. Note that before R_j is broadcast, R_j has been reduced by all the relations broadcast earlier. Therefore, the cardinality of R_j is

$$\prod_{h=1}^{i-1} \rho_{q(h)}^{j} \left|R_j\right|$$

when R_j is broadcast. Accordingly, the total transmission cost of sending all relations to the assembly site can be formulated as:

$$TR(q) = \sum_{i=1}^{n}\left(\left(\prod_{j=1}^{i-1} \rho_{q(j)}^{q(i)} \cdot \left|R_{q(i)}\right| \cdot CJ_{q(i)}^{A}\right)\right).$$

An illustrative example of $TR(q)$ is given below.

EXAMPLE 1. Suppose there are three relations, R_1, R_2, and R_3, where $|R_1| = 10$, $|R_2| = 20$, and $|R_3| = 40$. $CJ_1^A = CJ_2^A = CJ_3^A = 1$. $\rho_1^2 = 0.6$, $\rho_1^3 = 0.4$, and $\rho_2^3 = 0.5$. Let the broadcasting order be $\{q(1), q(2), q(3)\} = \{1, 2, 3\}$. Then, $TR(q) = 10 + 0.6 \cdot 20 + 0.4 \cdot 0.5 \cdot 40 = 30$.

The optimization problem for using relation semijoins on broadcasting networks, RSO, can be formally defined as follows.

DEFINITION 8 (Problem RSO). *Suppose there are n relations, $R_1, R_2, \ldots, R_n$. Given $|R_i|$, CJ_i^A, ρ_i^j, $1 \le i, j \le n$, determine the optimal permutation q for the broadcasting order such that TR(q) is minimized.*

An outline of the reduction from SPO to RSO is described first, and a formal problem reduction follows.

Description of reduction from SPO to RSO:

Problem setup: There are $n + 1$ relations, $R_1, R_2, \ldots R_{n+1}$, where R_{n+1} is a relation with very small selectivity, i.e., $\forall i, \rho_{n+1}^{i} \approx 0$. Let $|R_{n+1}| = 1$.

Decision: The ith decision, d_i, is set to 1 if R_i is broadcast before R_{n+1}. Otherwise, d_i is 0 and R_i is broadcast after R_{n+1}.

Default cost: If all d_i s are 0, R_{n+1} is the first relation to be broadcast. Since $\rho_{n+1}^{i} \approx 0$, the broadcasting cost for all the other relations is negligible. The total cost is approximated by the broadcasting cost of R_{n+1}, which is assumed to be one in the problem setup.

Extra cost: If d_i is 1, i.e., R_i is broadcast before R_{n+1}, the broadcasting cost of R_i will be $CJ_i^A \cdot |R_i|$, which is set to be s_i. The broadcasting cost of R_{n+1} will then be reduced by ρ_i^{n+1}, which is set to be p_i.

Difficulty: Broadcasting R_i before R_{n+1} reduces the broadcasting cost of R_{n+1}. However, an extra cost of transmitting R_i is incurred. It is difficult to optimize the trade-off between the reduction effect and the extra transmission cost.

DEFINITION 9 (Polynomial reduction from SPO to RSO). *Given an instance of SPO, we construct an instance of RSO with the following parameters:*

1) $|R_i| = s_i$, $CJ_i^A = 1, 1 \le i \le n$.
2) $|R_{n+1}| = CJ_{n+1}^A = 1$.
3) Let Δ be defined as in Definition 7. The selectivities are defined in the following table:

$\rho_i^j =$

$i \backslash j$	1	2	...	n	n + 1
1	-	1	...	1	p_1
2	1	-	...	1	p_2
:	:	:	:	:	:
n	1	1	...	-	p_n
n + 1	$\frac{1}{n+1}\Delta$	$\frac{1}{n+1}\Delta$	...	$\frac{1}{n+1}\Delta$	-

From Definition 9, we have the following lemma which proves RSO is NP-hard.

LEMMA 6.. *Problem RSO is NP-hard.*

PROOF. We shall prove that Definition 9 is a polynomial reduction from SPO to RSO, and this lemma follows. Clearly this mapping can be done in polynomial time. Next, define $B = \{i \mid q^{-1}(i) < q^{-1}(n+1)\}$, where q^{-1} is the inverse function of q. From Definition 9, we have,

$$\begin{aligned} TR(q) &= SP(B) + \frac{1}{n+1}\Delta \cdot \sum_{i=q^{-1}(n+1)+1}^{n+1} s_{q(i)} \\ &< SP(B) + \Delta. \end{aligned} \quad (4)$$

Following the same reasoning as in Lemma 5, it can be proved that $\exists B$, $SP(B) < c \Leftrightarrow \exists q$, $TR(q) < c$, leading to this lemma. □

4.4 Semijoin Optimization for Tree Queries

It has been shown that as far as optimizing the effect of semijoins is concerned, tree queries are fundamentally easier than cyclic queries [3], [18], [19]. In this section, we shall prove that optimization on the semijoin application for tree queries, though less complicated than that for cyclic queries, is still computationally difficult, and precisely, NP-hard.

A relation is said to be fully reduced if, after the execution of a sequence of semijoins, no other semijoins can further reduce its cardinality. A semijoin program is called optimal if it fully reduces the relations required in the join phase with the minimal transmission cost incurred. We shall consider determining the optimal semijoin programs for the following two types of tree queries:

1) **Single-reducer tree queries** [7], [13], [40]: After the semijoin reduction phase, only one relation is needed for final processing. The goal is to find an optimal semijoin program to fully reduce this relation.
2) **Full-reducer tree queries** [13], [19], [29], [40]: After the semijoin reduction phase, all relations are needed for final processing. The goal is to find an optimal semijoin program to fully reduce all relations.

4.4.1 Star Queries (Problem SQO)

To facilitate our discussion, we shall consider the star query, a special case of tree queries, first. A query is called a star query if its join graph is a star, i.e., there exists a relation R_0 such that every join specified in the query has R_0 as one of the operand relations. We shall use $(R_0, R_1, \ldots, R_n)$ to represent a star query with a central relation R_0, and leaf relations R_i, $1 \le i \le n$. Also, to simplify our presentation, same as in prior studies [7], [35], we assume that

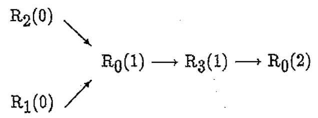
1) R_0 contains all the attributes in *TL*,
2) the assembly site is the one containing R_0, and
3) $a_{t(i)}$, the join attribute between R_0 and R_i, is a candidate key of both R_0 and R_i, for $1 \le i \le n$.

In the following discussion, we shall first present the results in [7], which, by dealing with SQO as a mathematical programming problem, proved that the optimal solution of SQO must belong to a special class of semijoin programs, thus greatly reducing the solution space of SQO. In light of this, we derive in Section 4.4.2 a polynomial reduction from SPO to SQO and prove that SQO is NP-hard.

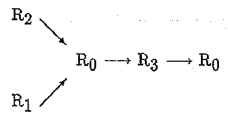
To describe the class of candidate semijoin programs for star queries, we introduce the notation of an execution graph, $GE(V_1, E_1)$, where V_1 is the set of instances of relations and E_1 is the set of semijoins. An instance of relation R_i is denoted by $R_i(j)$, where j is an integer representing the version number of R_i. An edge $R_i(m) \rightarrow R_j(k)$ in E_1 describes a semijoin from the *m*th version of R_i to the *k*th version of R_j. The 0th version of R_j, $R_j(0)$, refers to the original relation R_j before any semijoin reduction. The *k*th version of R_j refers to the relation R_j after all semijoins corresponding to incoming edges of $R_j(k)$ have been executed. An example execution graph is given in Fig. 4a, where $R_0 \ltimes R_2$ and $R_0 \ltimes R_1$ are first executed, followed by $R_3 \ltimes R_0$, and then $R_0 \ltimes R_3$. For clarity, the version numbers in an execution graph can be omitted, as in Fig. 4b, if there is no confusion. Then, as derived in [7], the execution graph of the optimal semijoin program for a star query has the following property.

THEOREM 2 [7]. *Given a star query $(R_0, R_1, \ldots, R_n)$, an optimal semijoin program can be described by (S, g), where $S = \{s(1), s(2), \ldots, s(k)\}$ is a subset of $\{1, 2, \ldots, n\}$, and g is a permutation function from $\{1, 2, \ldots, n - |S|\}$ to $\{1, 2, \ldots, n\} - S$. The execution graph of the optimal semijoin program exactly has the following edges as shown in* Fig. 5:

1) $\forall j \in S, R_j(0) \rightarrow R_0(1)$,
2) $R_0(h) \rightarrow R_{g(h)}(1)$ *and* $R_{g(h)}(1) \rightarrow R_0(h+1), 1 \le h \le n - |S|$.


$$\begin{tikzcd}
R_2(0) \arrow[dr] & & & \\
& R_0(1) \arrow[r] & R_3(1) \arrow[r] & R_0(2) \\
R_1(0) \arrow[ur] & & &
\end{tikzcd}$$

(a) The execution graph with version numbers.


$$\begin{tikzcd}
R_2 \arrow[dr] & & & \\
& R_0 \arrow[r] & R_3 \arrow[r] & R_0 \\
R_1 \arrow[ur] & & &
\end{tikzcd}$$

(b) The execution graph without version numbers.

Fig. 4. An example semijoin program.

PROOF. In order to fully reduce R_0, $R_0 \ltimes R_i$ has to be included in the semijoin program. However, $R_0 \ltimes R_i$ should be included at most once to guarantee the optimality of the semijoin program. Then, we have the following two cases, which lead to the edges determined in this theorem.

1) If $R_i \ltimes R_0$ is not included in the optimal semijoin program, then $R_0 \ltimes R_i$ should be executed before any semijoin $R_j \ltimes R_0$ is executed, since it is profitable to reduce $|R_0|$ before sending R_0 for other semijoins.
2) If $R_i \ltimes R_0$ is also included in the optimal semijoin program, then $R_0 \ltimes R_i$ should be scheduled immediately following $R_i \ltimes R_0$, since, same as in the first case, $|R_0|$ should be reduced first before executing other semijoins. □

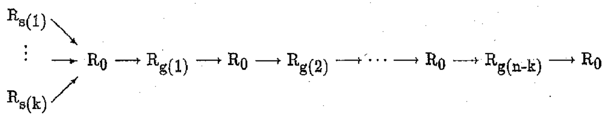

$$\begin{tikzcd}
R_{s(1)} \arrow[dr] & & & & & & & & & \\
\vdots \arrow[r] & R_0 \arrow[r] & R_{g(1)} \arrow[r] & R_0 \arrow[r] & R_{g(2)} \arrow[r] & \cdots \arrow[r] & R_0 \arrow[r] & R_{g(n-k)} \arrow[r] & R_0 \\
R_{s(k)} \arrow[ur] & & & & & & & & &
\end{tikzcd}$$

Fig. 5. An execution graph for a star query.

S, $\{1, 2, \ldots, n\} - S$, and $g()$, defined in Theorem 2, are called, respectively, the parallel program, the serial program, and the ordering function. Theorem 2 reduces the star query processing problem to the one of determining the optimal parallel program and the ordering function.

We next discuss the cost of the semijoins. If $R_0 \ltimes R_i$ is in the parallel program, its cost is $CS_i^0|R_i|$. On the other hand, if $R_i \ltimes R_0$ is in the serial program, its cost is $CS_i^0 \cdot X$, where X is the cardinality of R_0 right before this semijoin is executed. The cost of $R_0 \ltimes R_i$ in the serial program can be expressed as $CS_i^0 \cdot Y$, where Y is the cardinality of R_i after being reduced by $R_i \ltimes R_0$. Since $a_t(i)$ is a candidate key for both R_0 and R_i, we have

$$Y = \left|R_0.a_{t(i)} \cap R_i.a_{t(t)}\right| = \rho_i^0 \left|R_0.a_{t(i)}\right| = \rho_i^0 \cdot X.$$

Therefore, $R_0 \ltimes R_i$ costs $CS_i^0 \cdot \rho_i^0 \cdot X$.

Based on the foregoing, the cost of the semijoin program, denoted by $TS(S, g)$, can be expressed as:

$$TS(S,g) = \sum_{i \in S} \left(CS_i^0 |R_i|\right) + \sum_{j=1}^{n-|S|} \left(\prod_{k \in S} \rho_k^0 \cdot \prod_{h=1}^{j-1} \rho_{g(h)}^0 \cdot |R_0| \cdot \left(CS_0^{g(j)} + \rho_{g(j)}^0 \cdot CS_{g(j)}^0 \right) \right).$$

An illustrative example of $TS(S, q)$ is given below.

EXAMPLE 2. Consider a star query (R_0, R_1, R_2, R_3), where $|R_0| = 100$, $|R_1| = 10$, $|R_2| = 20$, and $|R_3| = 30$. $\rho_1^0 = 0.5$, $\rho_2^0 = 0.2$, and $\rho_3^0 = 0.1$. $\forall i$, $CS_0^i = CS_i^0 = 1$. The execution graph in Fig. 4a costs $10 + 20 + 0.5 \cdot 0.2 \cdot 100 \cdot (1 + 0.1) = 41$.

The star query optimization problem SQO can be formally defined as follows.

DEFINITION 10 (Problem SQO). *Given a star query (R_0, R_1, ..., R_n), determine an optimal semijoin program (S, g) such that $TS(S, g)$ is minimized.*

4.4.2 Single-Reducer Tree Queries (Problem SSQO)

Notice that SQO is a subproblem of single-reducer tree queries because a star graph is a special case of tree graphs, and also R_0 is the only relation that needs to be fully reduced. We shall prove SQO is NP-hard, and the NP-hardness of SSQO thus follows. As before, a reduction from SPO to SQO is outlined first.

Description of reduction from SPO to SQO:

Problem setup: We create a star query with $n + 1$ leaf relations. R_{n+1} is a special relation with very large cardinality and very small selectivity. Since R_{n+1} has a large cardinality, it has to be scheduled in the serial program; otherwise, $R_0 \ltimes R_{n+1}$ is too costly. Since R_{n+1} has a very small selectivity, it is always beneficial to let R_{n+1} be the first relation in the serial program, i.e., $g(1) = n + 1$. By doing so the cardinality of R_0 can be significantly reduced, and the remaining serial program thus has a negligible cost.

Decision: The ith decision, d_i, is set to 1 if R_i is included in the parallel program. Otherwise, d_i is 0 and R_i is included in the serial program.

Default cost: If all d_is are 0, then $R_{n+1} \ltimes R_0$ (i.e., the first semijoin in the serial program) costs one. Every other semijoin in the serial program has a negligible cost.

Extra cost: If d_i is 1, the cost of $R_0 \ltimes R_i$ is set to be s_i. Then, the cost of $R_{n+1} \ltimes R_0$ is reduced by a factor of ρ_i^0. ρ_i^0 is set to be p_i in our construction.

Difficulty: It is difficult to determine whether $R_0 \ltimes R_i$ should be included in the parallel program or in the serial program.

Formally, the polynomial reduction from SPO to SQO is described below.

DEFINITION 11 (Polynomial reduction from SPO to SQO). *For a given instance of SPO, we create a star query of $n + 1$ leaf relations with the following parameters:*

1) $|R_0| = 1$.
2) $CS_i^0 = CS_0^i = 1, 1 \le i \le n + 1$.
3) $|R_i| = s_i$, $\rho_i^0 = p_i$, $1 \le i \le n$.
4) $|R_{n+1}| = 2$.
5) $\rho_{n+1}^0 = \frac{1}{2(n+1)} \Delta$, *where Δ is defined in Definition 7.*

LEMMA 7. *For any star query derived from Definition 11, if (S, g) is optimal, then $g(1) = n + 1$, i.e., R_{n+1} must be the first relation in the serial program.*

PROOF. The proof consists of two parts:

1) $n + 1 \notin S$, and
2) $g^{-1}(n + 1) = 1$.

Suppose there is an optimal semijoin program (S_1, g_1) such that $n + 1 \in S_1$. Define

$$g_2(i) = \begin{cases} n+1, & \text{if } i = 1, \\ g_1(i-1), & \text{otherwise.} \end{cases}$$

It can be verified that ($S_1 - \{n + 1\}$, g_2) always costs less than (S_1, g_1). Clearly, $R_0 \ltimes R_{n+1}$ costs 2 if it is included in the parallel program. If we move R_{n+1} to the first position in the serial program, then $R_{n+1} \ltimes R_0$ and $R_0 \ltimes R_{n+1}$ cost no more than $1 + \rho_{n+1}^0$, which is less than 2. Note that such a movement will not change the costs of other semijoins. Thus the new program costs less than (S_1, g_1), leading to a contradiction.

It remains to prove that $g^{-1}(n + 1) = 1$. We shall again prove this equality by contradiction. Suppose there is an optimal semijoin program (S_3, g_3) such that $g_3^{-1}(n+1) = k > 1$. Use q to denote $g_3(k - 1)$ for simplicity. Define g_4 as g_3 except $g_4(k - 1) = n + 1$ and $g_4(k) = q$. We claim the cost of (S_3, g_4) is less than that of (S_3, g_3). Consider the execution graph corresponding to (S_3, g_3) first. It can be obtained that the semijoin subprogram $R_0 \to R_q \to R_0 \to R_{n+1} \to R_0$ costs

$$COST1 = \left(CS_0^q + \rho_q^0 \cdot CS_q^0 + \rho_q^0 \cdot CS_0^{n+1} + \rho_q^0 \cdot \rho_{n+1}^0 \cdot CS_{n+1}^0\right) \cdot X,$$

where

$$X = \left(\prod_{k \in S_3} \rho_k^0 \cdot \prod_{h=1}^{k-1} \rho_{g_3(h)}^0 \cdot |R_0| \right),$$

i.e., the cardinality of R_0 right before $R_q \ltimes R_0$ is executed.

Similarly, for (S_3, g_4), the cost of $R_0 \to R_{n+1} \to R_0 \to R_q \to R_0$ can be expressed as,

$$COST2 = \left(CS_0^{n+1} + \rho_{n+1}^0 \cdot CS_{n+1}^0 + \rho_{n+1}^0 \cdot CS_0^q + \rho_{n+1}^0 \cdot \rho_q^0 \cdot CS_q^0\right) \cdot X.$$

$$\begin{tikzcd}
R_{s(1)} \arrow[dr] & & & & & & & R_1 & \\
\vdots \arrow[r] & R_0 \arrow[r] & R_{g(1)} \arrow[r] & R_0 \arrow[r] & \cdots \arrow[r] & R_{g(n+1-|S|)} \arrow[r] & R_0 \arrow[r] \arrow[ur] \arrow[dr] & \vdots & \left.\right\} \text{except } R_{g(n+1-|S|)} \\
R_{s(k)} \arrow[ur] & & & & & & & R_{n+1} &
\end{tikzcd}$$

Fig. 6. The full reducer constructed in Lemma 10.

From Definition 11, we have $COST2 < COST1$ because $CS_0^q = CS_q^0 = CS_0^{n+1} = CS_{n+1}^0 = 1$ and $\rho_{n+1}^0 < \rho_q^0$. Also, it can be verified that the costs of other semijoins remain the same in both (S_3, g_3) and (S_3, g_4). Thus, the new semijoin program (S_3, g_4) costs less than (S_3, g_3), contradicting to the optimality of (S_3, g_3). This lemma follows. □

We then have the following lemma which proves SQO is NP-hard.

LEMMA 8. *Problem SQO is NP-hard.*

PROOF. We shall prove that Definition 11 is a polynomial reduction from SPO to SQO and this lemma follows. It can be seen that the reduction in Definition 11 can be done in polynomial time. Let $(R_0, R_1, \ldots, R_{n+1})$ be the star query constructed in Definition 11, and (S, g) be the optimal semijoin program. From Lemma 7, $S \subset \{1, 2, \ldots, n\}$ and $g(1) = n + 1$. Thus,

$$TS(S,g) = \sum_{i \in S} s_i + \prod_{i \in S} p_i \cdot \left(1 + \frac{1}{2n+2}\Delta + \frac{1}{2n+2}\Delta \cdot \sum_{j=2}^{n+1-|S|}\left(\prod_{h=2}^{j-1} p_{g(h)} \cdot \left(1 + p_{g(j)}\right)\right)\right)$$
$$< SP(S) + \Delta.$$

It can be observed that there are at most $2n + 1$ semijoins executed after $R_{n+1} \ltimes R_0$ and each of them costs no more than $\frac{1}{2n+2} \cdot \Delta$. Thus, the total cost of all semijoins after $R_{n+1} \ltimes R_0$ is less than Δ.

Following the same reasoning as in Lemma 5, it can be proved that $\exists B, SP(B) < c \Leftrightarrow \exists (S, g), TS(S, g) < c$, leading to this lemma. □

Lemma 8 leads to the following lemma since a star query is a special case of tree queries.

LEMMA 9. *Problem SSQO is NP-hard.*

4.4.3 Full-Reducer Tree Queries (Problem FSQO)

We now consider the full-reducer star query processing, which requires that the leaf relations, as well as the central relation, be fully reduced. Notice that any full reducer EX for a star query can be partitioned into two subprograms:

1) EX_c: a single reducer for R_0, and
2) EX_l: the semijoin program to reduce the leaf relations if they have not been fully reduced by EX_c yet.

Clearly, EX_c is a candidate solution to SSQO. In this section, we shall construct a full-reducer star query optimization problem in such a way that the total cost is dominated by that of EX_c. The NP-hardness of FSQO thus follows.

DEFINITION 12 (Polynomial reduction from SPO to FSQO). *For a given instance of SPO, we create a star query of $n + 1$ leaf relations with the following parameters:*

1) $|R_0| = 1$.
2) $CS_i^0 = CS_0^i = 1, 1 \le i \le n + 1$.
3) $|R_i| = s_i, \rho_i^0 = p_i, 1 \le i \le n$.
4) $|R_{n+1}| = 2$.
5) $\rho_{n+1}^0 = \frac{1}{3n+2}\Delta$, *where Δ is defined in Definition 7.*

The following lemma, stating FSQO is NP-hard, then follows.

LEMMA 10. *Problem FSQO is NP-hard.*

PROOF. Let (S, g) be the optimal single reducer for the constructed star query. Following the same reasoning as in Lemma 7, we have $g(1) = n + 1$. Since there are at most $2n + 1$ semijoins executed after $R_{n+1} \ltimes R_0$, and each of them costs less than $\frac{1}{3n+2}\Delta$. Thus,

$$TS(S,g) =$$
$$SP(S) + \frac{1}{3n+2}\Delta \cdot \prod_{k \in S} p_k \left(1 + \sum_{j=2}^{n+1-|S|}\left(\prod_{h=2}^{j-1} p_{g(h)} \cdot \left(1 + p_{g(j)}\right)\right)\right) \quad (5)$$
$$< SP(S) + \frac{2n+1}{3n+2}\Delta. \quad (6)$$

Analogous to the derivation for Lemma 8, it can be shown that $\exists B, SP(B) < c \Leftrightarrow TS(S, g) < c$. Therefore, it suffices to prove that $TS(S,g) < c \Leftrightarrow \exists$ a full reducer EX, $COST(EX) < c$.

Suppose $TS(S, g) \ge c$. Since (S, g) is the optimal single-reducer for R_0, for any full reducer $EX = EX_c \cup EX_l$, we have $\mathrm{COST}(EX_c) \ge c$. Consequently, $\mathrm{COST}(EX) \ge \mathrm{COST}(EX_c) \ge c$.

To prove the other direction, construct a full reducer $EX' = EX_c' \cup EX_l'$, where $EX_c' - (S, g)$ and the execution graph of EX_l' contains the following edges:

$$\{R_0(n+2-|S|) \to R_i(1) \mid 1 \le i \le n+1 \text{ and } i \in S\} \cup$$
$$\{R_0(n+2-|S|) \to R_i(2) \mid 1 \le i \le n+1, i \notin S, \text{ and } i \ne g(n+1-|S|)\}.$$

The execution graph of EX' is shown in Fig. 6, where $\{s(1), s(2), \ldots, s(k)\}$ represents S.

There are n semijoins in EX_l' and each of them costs less than $\frac{1}{3n+2}\Delta$. Therefore, $COST(EX_l') < \frac{n}{3n+2}\Delta$. Then, from (6), we have,

$$CSOT(EX') = TS(S,g) + COST(EX_l')$$
$$< SP(S) + \Delta.$$

From (5), $TS(S, g) < c$ implies $SP(S) < c$. Then, following the same reasoning as in Lemma 5, we have $COST(EX') < c$, thus proving this lemma. □

5 Conclusions

In this paper we generally characterized the distributed query optimization problems and provided a frame work to explore their complexity. It has been shown that most distributed query optimization problems can be transformed into the Sum Product Optimization problem. We first proved SPO is NP-hard. Then, using this result as a basis, we proved that the following five classes of distributed query optimization problems:

1) local optimization of semijoins,
2) join sequence optimization,
3) relation semijoints on broadcasting networks,
4) single-reducer tree queries, and
5) full-reducer tree queries,

which cover the majority of distributed query optimization problems previously studied in the literature, are NP-hard by polynomially reducing SPO to each of them. The detail for each problem transformation has been derived. We not only proved the conjecture that many prior studies relied upon, but also provide a frame work for future related studies. Note that the usefulness of NP-hardness of SPO proved in this paper is not confined to the context of distributed query optimization, and is in fact applicable to solving the complexity of many other optimization problems.

Appendix
List of Symbols

SPO;	Sum product optimization problem.
SPD:	Sum product optimization decision problem.
KNAP:	Knapsack problem.
LO:	Local optimization of semijoins.
JSO:	Join query optimization problem.
RSO:	Relation semijoin optimization on broadcasting networks.
SQO:	Star queries optimization problem.
SSQO:	Single-reducer tree query optimization.
FSQO:	Full-reducer tree query optimization.

Acknowledgments

The authors are grateful for S. Shyu at Doumentum and J. Chen at IBM for their helpful comments on improving this paper.

References

[1] P.M.G. Apers, A.R. Hevner, and S.B. Yao, "Optimization Algorithms for Distributed Queries," *IEEE Trans. Software Eng.*, vol. 9, no. 1, pp. 57-68, Jan. 1983.

[2] P.A. Bernstein, N. Goodman, E. Wong, C. Reeve, and J.B. Rothnie, "Query Processing in a System for Distributed Databases (SDD-1)," *ACM Trans. Database Systems*, vol. 6, no. 4, pp. 602-625, Dec. 1981.

[3] P.A. Bernstein and D.M. Chiu, "Using Semi-Joins to Solve Relational Queries," *J. ACM*, vol. 28, no. 1, pp. 25-40, Jan. 1981.

[4] P.A. Black and W.S. Luk, "A New Heuristic for Generating Semi-Join Programs for Distributed Query Processing," *Proc. IEEE COMPSAC*, pp. 581-588, 1982.

[5] S. Ceri and G. Pelagatti, *Distributed Databases Principle and Systems.* New York: McGraw-Hill, 1985.

[6] A.L.P. Chen and V.O.K. Li, "Improvement Algorithms for Semi-Join Query Processing Programs in Distributed Database Systems," *IEEE Trans. Computers*, vol. 33, no. 11, pp. 959-967, Nov. 1984.

[7] A.L.P. Chen and V.O.K. Li, "An Optimal Algorithm for Distributed Star Queries," *IEEE Trans. Software Eng.*, vol. 11, no. 10, pp. 1,097-1,107, Oct. 1985.

[8] J.S.J. Chen and V.O.K. Li, "Optimizing Joins in Fragmented Database Systems on a Broadcast Local Network," *IEEE Trans. Software Eng.*, vol. 15, no. 1, pp. 26-38, Jan. 1989.

[9] M.-S. Chen and P.S. Yu, "Interleaving a Join Sequence with Semijoins in Distributed Query Processing," *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 5, pp. 611-621, Sept. 1992.

[10] M.-S. Chen and P.S. Yu, "Combining Join and Semijoin Operations for Distributed Query Processing," *IEEE Trans. Knowledge and Data Eng.*, vol. 5, no. 3, pp. 534-542, June 1993.

[11] M.-S. Chen and P.S. Yu, "A Graph Theoretical Approach to Determine a Join Reducer Sequence in Distributed Query Processing," *IEEE Trans. Knowledge and Data Eng.*, vol. 6, no. 1, pp. 152-165, Feb. 1994.

[12] D.M. Chiu, P.A. Bernstein, and Y.C. Ho, "Optimizing Chain Queries in A Distributed Database System," *SIAM J. Computing*, vol. 13, no. 1, pp. 116-134, Feb. 1984.

[13] D.M. Chiu and Y.C. Ho, "A Methodology for Interpreting Tree Queries into Optimal Semi-Join Expressions," *Proc. ACM SIGMOD*, pp. 169-178, May 1980.

[14] W.W. Chu and P. Hurley, "Optimal Query Processing for Distributed Database Systems," *IEEE Trans. Computers*, vol. 31, no. 9, pp. 135-150, Sept. 1982.

[15] D. Gardy and C. Puech, "On the Effect of Join Operations on Relation Sizes," *ACM Trans. Database Systems*, vol. 14, no. 4, pp. 574-603, Dec. 1989.

[16] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[17] B. Gavish and A. Segev, "Set Query Optimization in Distributed Database Systems," *ACM Trans. Database Systems*, vol. 11, no. 3, pp. 266-293, Sept. 1986.

[18] N. Goodman and O. Shmueli, "The Tree Property Is Fundamental for Query Processing," *Proc. ACM Symp. Principles of Database Systems*, pp. 40-48, 1982.

[19] N. Goodman and O. Shmueli, "Tree Queries: A Simple Class of Relational Queries," *ACM Trans. Database Systems*, vol. 7, no. 4, pp. 653-677, Dec. 1982.

[20] A.R. Hevner, "The Optimization of Query Processing on Distributed Database Systems," PhD thesis, Purdue Univ., 1979.

[21] A.R. Hevner, O.Q. Wu, and S.B. Yao, "Query Optimization on Local Area Networks," *ACM Trans. Office Information*, vol. 3, pp. 35-62, Jan. 1985.

[22] A.R. Hevner and S.B Yao, "Query Processing in Distributed Databases," *IEEE Trans. Software Eng.*, vol. 5, no. 3, pp. 177-187, May 1979.

[23] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms.* Computer Science Press, 1978.

[24] K.T. Huang, "Query Optimization in Distributed Databases," PhD thesis, Laboratory for Information and Decision Systems, Massachusetts Inst. of Technology, 1982.

[25] Y. Kambayashi, M. Yoshikawa, and S. Yajima, "Query Processing for Distributed Databases Using Generalized Semi-Joins," *Proc. ACM SIGMOD*, pp. 151-160, 1982.

[26] H. Kang and N. Roussopoulos, "Combining Joins and Semijoins in Distributed Query Processing," Technical Report CS-TR-1794, Computer Science Dept., Univ. of Maryland, College Park, 1987.

[27] S. Lafortune and E. Wong, "A State Transition Model for Distributed Query Processing," *ACM Trans. Database Systems*, vol. 11, no. 3, pp. 294-322, Sept. 1986.

[28] W. Perrizo, J.Y. Li, and W. Hoffman, "Algorithms for Distributed Query Processing in Broadcasting Local Area Networks," *IEEE Trans. Knowledge and Data Eng.*, vol. 1, no. 2, pp. 215-225, June 1989.

[29] S. Pramanik and D. Vineyard, "Optimizing Join Queries in Distributed Databases," *IEEE Trans. Software Eng.*, vol. 14, no. 9, pp. 1,319-1,326, Sept. 1988.

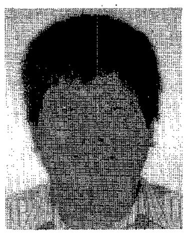
[30] G. Sacco, "Distributed Query Evaluation in Local Area Networks," *Proc. IEEE Data Eng. Conf.*, pp. 510-516, Apr. 1984.

[31] Arie Segev, "Global Heuristics for Distributed Query Optimization," *Proc. IEEE INFOCOM*, pp. 388-394, Apr. 1986.

[32] D. Shasha and T.-L. Wang, "Optimizing Equijoin Queries in Distributed Databases Where Relations Are Hash Partitioned," *ACM Trans. Database Systems*, vol. 16, no. 2, pp. 279-308, June 1991.

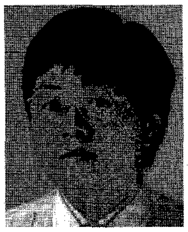
[33] W. Sun and C. Yu, "Semantic Query Optimization for Tree and Chain Queries," *IEEE Trans. Knowledge and Data Eng.*, vol. 6, no. 1, pp. 136-151, Feb. 1994.

[34] P. Valduriez and G. Gardarin, "Join and Semijoin Algorithms for a Multiprocessor Database Machine," *ACM Trans. Database Systems*, vol. 9, no. 1, pp. 133-161, Mar. 1984.
[35] C.P. Wang and V.O.K. Li, "The Relation-Partitioning Approach to Distributed Query Processing," *Proc. Second IEEE Data Eng. Conf.*, pp. 21-28, Feb. 1986.
[36] C.P. Wang, V.O.K. Li, and A.L.P. Chen, "One-Shot Semi-Join Execution Strategies for Processing Distributed Queries," *Proc. Seventh IEEE Data Eng. Conf.*, pp. 756-763, Apr. 1991.
[37] E. Wong, "Retrieving Dispersed Data from SDD-1: A System for Distributed Databases," *Proc. Second Berkeley Workshop Distributed Data Management and Computer Networks*, pp. 217-235, May 1977.
[38] H. Yoo and S. Lafortune, "An Intelligent Search Method for Query Optimization by Semijoins," *IEEE Trans. Knowledge and Data Eng.*, vol. 1, no. 2, pp. 226-237, June 1989.
[39] C. Yu and C.C. Chang, "Distributed Query Processing," *ACM Computing Surveys*, vol. 16, no. 4, pp. 399-433, Dec. 1984.
[40] C. Yu, Z.M. Ozsoyoglu, and K. Kam, "Optimization of Distributed Tree Queries," *J. Comput. Syst. Sci.*, vol. 29, no. 3, pp. 409-445, Dec. 1984.



Chihping Wang received the BSEE degree in electrical engineering from the National Taiwan University, Taiwan, in 1983, and the PhD degree in computer engineering from the University of Southern California, Los Angeles, in 1988.

From 1988 to 1993, he was an assistant professor of computer science at the University of California, Riverside. He is currently with Informix Software, Inc. His research interests include distributed database systems, object-oriented databases, and computer networks. Dr. Wang is a member of the IEEE.



Ming-Syan Chen (S'87-M'88-SM'93) received the BS degree in electrical engineering from National Taiwan University, Taipei, Taiwan in 1982, and the MS and PhD degrees in computer, information, and control engineering from the University of Michigan, Ann Arbor, in 1985 and 1988, respectively.

He was a research staff member at IBM T.J. Watson Research Center, Yorktown Heights, New York, from 1988 to 1996, involved in issues related to database mining and digital libraries. He is currently a member of the faculty of the Electrical Engineering Department at National Taiwan University, Taipei, Taiwan. From 1985 to 1988, he was a member of the Real-Time Computing Laboratory at the University of Michigan, Ann Arbor, where he conducted research on fault-tolerant routing and processor allocation for hypercube multicomputers. His research interests include parallel databases, multimedia systems, Internet applications, and graph and combinatorial theory.

Dr. Chen has published more than 70 refereed journal/conference papers in his research areas. He holds 15 patents in the areas of interactive video playout, video server design, interconnection network, and concurrency and coherency control protocols, and is also a conference tutorial lecturer on parallel query processing. He received the Outstanding Innovation Award from IBM in 1994 for his contribution to parallel transaction design for a major database product, and numerous awards for his inventions and patent applications. Dr. Chen is a member of the Association for Computing Machinery.