中華民國八十四年全國計算機會議論文集
Proceedings of National Computer Symposium 1995

一個以目標關係為基礎的的分散聯結運算方式
A Target-Relation-Based Approach to Distributed Joins*

張玉盈，劉博敏
Ye-In Chang and Bor-Miin Liu

國立中山大學應用數學系
Dept. of Applied Mathematics
National Sun Yat-Sen University
Kaohsiung, Taiwan
R.O.C.
{E-mail: changyi@math.nsysu.edu.tw}
{Tel: 886-7-5316171 (ext. 3710)}
{Fax: 886-7-5319475}

摘要

所謂目標關係(target relations)就是對於使用者所感興趣的資料所在。在這篇論文中，藉著指出目標關係，我們將一個查詢圖分成一個決定性的查詢圖和零個或一個以上的非決定性的查詢圖。對於這些非決定性查詢圖，我們所要做的是去推導出一串最佳的半聯結運算去完全降低這個根節點。因此，當我們有超過一個以上的非決定性查詢圖時，我們可以採取平行的方式來處理，以縮短查詢反應時間。接著，我們同時利用聯結與半聯結運算，使決定性查詢圖得到最大的利益。所以我們以目標關係為基礎的方法，不僅能降低資料傳送成本，而且也可以縮短反應時間。如果非決定性查詢圖的數目越多，則我們能降低更多的資料傳送成本。最後，我們將會看到，和其他半聯結運算在聯結運算之前的方法，或是同時運用半聯結與聯結運算的方法相較，此論文所提之方法可得更高之效率。

(關鍵詞：分散式資料庫，啓發式演算法，查詢最佳化，關係式資料庫，半聯結)

Abstract

A *target relation* of a query is a relation which contains attributes of selected tuples to be outputed. In this paper, by identifying target relations, we divide a given tree query into two parts: one *final query tree* and zero or more *non-final query trees*. Since only the root of each of the *non-final query trees* will participate in the *final query tree*, we can apply a semijoin program to fully reduce the size of the root of each of the *non-final trees* first. Therefore, we can reduce the data transmission cost for the final query tree. Moreover, when there is more than one non-final query trees, we can process them in parallel, which can shorten the query response time. Then, we apply join and semijoin operations together to optimize the the cost of the *final query tree*. Consequently, our target-relation-based approach not only can reduce the data transmission cost but also the response time. Moreover, the larger the number of non-final query trees is, the more reduction our approach can achieve. We show that the proposed approach to distributed joins can have better performance than other approaches which either apply semijoins before the join process or apply both joins and semijoins together.

(Key Words: distributed databases, heuristic joins, query optimization, relational databases, semijoins)

1 Introduction

In a distributed database, we have the ability to decentralized data that are most heavily used by end users at geographically dispread locations and, at the same time, to combine data from different sources by means of queries. In a distributed relational database system, the processing of a query involves data transmission among different sites (or nodes) via a communication network. The retrieval of data from different sites in a network is known as *distributed query processing*. The problem of optimal query processing in distributed database systems was shown to be NP-hard, where an optimal query processing program is one which requires the least total data transmission cost to process the query.

In a wide area network, under the assumptions that each site contains one relation, there is only

* This research was supported by the National Science Council of the Republic of China, Grant No. NSC-83-0408-E-110-013

one copy of each relation, and the cost of local processing is negligible compared to the transmission cost, a query is usually processed in the following three phases [6]: (1) *local processing phase* which involves all local processing such as selections and projections, (2) *reduction phase* where a sequence of *semijoins* is used to reduce the size of relations, and (3) *final processing phase* in which all resulting relations are sent to the site where the final query processing is performed. The *semijoin* operation have been extensively studied in the literature [3, 4, 17]. The *semijoin* operator takes the join of two relations, R and S, and then projects back out on the domains of relation R. For the semijoins to be performed, only the projection of the joining attribute need be sent. If the size of these projections is small relative to the amount by which R and S are reduced, then the preliminary semijoin will be *profitable*.

The first algorithm using semijoins for distributed query processing was implemented in SDD-1 in [3]. This SDD-1 algorithm is based on a *hill-climbing* strategy that produces efficient, but not necessarily optimal query processing strategies. Theoretical aspects of semijoins were first studied in [2]. *Simple queries* were studied in [13]. Their algorithm for general queries was improved in [1]. It has been proved that a *tree query* can be fully reduced by using semijoins [2], and there has been much research reported in optimizing semijoin sequences to process certain tree queries, such as *star queries* [5] and *chain queries* [11]. However, the determination of an optimal semijoin sequence to process certain tree queries has been proved to be NP-hard [12]. For general query graphs with cycles, even with one join attribute, the problem of finding an optimal strategy to minimize the data transmission cost has also been proved to be NP-hard [12]. Methods based on *dynamic programming* to get an optimal semijoin sequence for *tree queries* and *chain queries*, were studied in [10] and [11], respectively. These methods based on dynamic programming have a high computational complexity which limits their applicability. A *heuristic* approach to finding a semijoin program that only fully reduces one relation is proposed in [17]. In [4], they describe algorithms to improve the solutions generated by heuristics.

In addition to semijoins, join operations can also be used as reducers in distributed query processing to further reduce the communication cost [6, 8]. Moreover, the approach of combining join and semijoin operations as reducers can result in more beneficial semijoins due to the inclusion of joins as reducer [8]. (Note that such semijoins are referred to as *gainful semijoins.*) That is, this approach considers both phases (2) and (3) together. Both *profitable semijoins* and *gainful semijoins* are called *beneficial semijoins*.

In this paper, different from those algorithms based on either only the profitable semijoins or the gainful semijoins to reduce the data transmission cost in phases (2) or/and (3), we propose a *target-relation-based* approach to distributed joins. A *target relation* of a query is a relation which contains attributes of selected tuples to be outputed. By identifying target relations, we divide a given tree query into two parts: one *final query tree* and zero or more *non-final query trees.* A *final query tree* contains *final relations* that are target relations and those relations which are intermediate nodes in the paths between any two target relations in the given tree query. A *non-final query tree* contains those relations which participate in *non-final joins*, where a *non-final join* means that at least one of its joining relations is not a final relation. Since only the root of each of the *non-final query trees* will participate in the *final query tree*, we can apply a semijoin program to fully reduce the size of the root of each of the *non-final trees* first. Therefore, we can reduce the data transmission cost for the final query tree. Moreover, when there is more than one non-final query trees, we can process them in parallel, which can shorten the query response time. Then, we apply join and semijoin operations together to optimize the the cost of the *final query tree*. Consequently, our target-relation-based approach not only can reduce the data transmission cost but also the response time. Moreover, the larger the number of non-final query trees is, the more reduction our approach can achieve. We show that the proposed approach distributed joins can have better performance than other approaches which either apply semijoins before the join process or apply both join and semijoin together.

The rest of the paper is organized as follows. In Section 2, we give some definitions used in this paper. In Section 3, we present our proposed strategy. Finally, in Section 4, we give a conclusion.

2 Background

In this section, we describe assumptions and definitions used in the paper.

2.1 Queries, Query Graphs, Joins and Semijoins

A query Q consists of two components: *the target list* and *the qualification*. The *qualification* component selects the tuples of the referenced relations that satisfy the qualification, while the *target* component specifies attributes of the selected tuples which are to be outputed to the users. Given a query Q with qualification q, we define its corresponding *query graph* $G_Q(V_Q,E_Q)$ as follows:

V_Q = {set of all relation names referenced by q };

$E_Q = \{(i,\ j) \mid i \neq j$ and some clause of q references both R_i and R_j $\}$.

Figure 1 shows the query graph for the following query, where $R_1.A$ is the target list:

A select $R_1.A$
from R_1, R_2, R_3
where $R_1.A = R_2.A$ and $R_2.B = R_3.B$.

We call a query a *tree query* either if its query graph is a tree or if it is equivalent to a query whose query graph is a tree. We assume that we know the following information about the relations.

For each relation R_i, i=1,2,...,m,
$|R_i|$: number of tuples;
w_{R_i}: size (e.g., in bytes) of R_i.
For each attribute A of relation R_i,

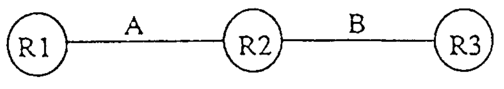

Figure 1: A query graph G_{EX1}

$|R_i(A)|$: cardinality;
$\rho_{i,A}$: selectivity;
$w_{R_i(A)}$: size (e.g., in bytes) of the data item in attribute A of relation R_i.

The *cardinality* of attribute A of relation R_i, denoted as $|A|$, is the number of distinct values in attribute A of relation R_i and the *selectivity* $\rho_{i,A}$ of attribute A is defined as the number of different values occurring in the attribute divided by the number of all possible values of the attribute.

A join clause "R_1 *joins* R_2 *on* A" is denoted by $R_1 \xleftrightarrow{A} R_2$, where R_1 and R_2 are relations, and attribute A is the joining attribute. Associated with this join are two semijoins: R_1 by R_2 on A, and R_2 by R_1 on A, denoted by $R_2 \xrightarrow{A} R_1$, and $R_1 \xrightarrow{A} R_2$, respectively. $R_1 \xrightarrow{A} R_2$ entails shipping $R_1(A)$, attribute A of R_1, to the site where R_2 resides and joining $R_1(A)$ with R_2. We denote the resulting relation by R_2' (and R_1 is unchanged).

2.2 Properties of Semijoins

We say that a relation R_i is *reduced* by a relation R_j in a semijoin program if the semijoin program has an *embedded chain* such that the head of the *chain* is R_j and the tail of the chain is R_i [16]. For example, in the semijoin program

$$R_1 \rightarrow R_2, R_1 \rightarrow R_3, R_4 \rightarrow R_5, R_3 \rightarrow R_6, R_5 \rightarrow R_7,$$
$$R_8 \rightarrow R_9,$$

the following semijoin programs are *embedded chains*:

$$(R_1 \rightarrow R_3, R_3 \rightarrow R_6) and (R_4 \rightarrow R_5, R_5 \rightarrow R_7).$$

A relation R_i is said to be *fully reduced* in a query graph if given any relation R_j in the query graph such that $i \neq j$, R_i is reduced by R_j. A full reducer program for a tree query is a semijoin program which reduces each relation in the tree query fully. An example of a full reducer program for the query graph shown in Figure 1 is illustrated as follows

$$R_1 \rightarrow R_2, R_3 \rightarrow R_2, R_2 \rightarrow R_3, R_2 \rightarrow R_1.$$

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

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

$$R_2 \rightarrow R_3, R_3 \rightarrow R_2, R_1 \rightarrow R_2,$$

is a single reducer program for relation R_2 of the query graph shown in Figure 1. One simplest way to derive a single reducer program based on semijoins is to include all *backward* semijoins in a breadth-first left-to-root order [2]. (Note that each edge of a rooted join tree has two directions, each corresponding to a semijoin. The one directed toward the root node is called a *backward* semijoins.) To derive a single reducer program, Pramanik et al. [16] have proposed a strategy based on the concept of a *minimal cover*, Yoo et al. [17] have applied a heuristic search (based on the A^* algorithm), and Chang et al. [14] have proposed an *Eulerian-path-like-based* strategy.

2.3 Cost and Benefit of Semijoin Reducers

In this paper, we concentrate on reducing the communication cost. We assume that the local processing cost has a negligible contribution to the total cost. Thus we need to consider only the cost of transmitting the data. We assume that the transmission cost is given by cost$(n) = c_0 + c_1 * n$, where n is the amount of data transmitted and c_0 and c_1 are constants. That is, we assume that data transmission cost is proportional to the volume of data to be transmitted. Let the transmission cost be one per data unit transmitted. Consider a semijoin $R_i \xrightarrow{A} R_j$ when R_i and R_j are at different sites. Then, the *cost* of the semijoin is

$$size\_of \ R_i[A],$$

and the *benefit* is

$$size\_of \ R_j \ before \ semijoin \ - \ size\_of \ R_j$$
$$after \ semijoin,$$

where the size of the relations is measured in bytes. A semijoin $R_i \xrightarrow{A} R_j$, is called *profitable* if its cost of sending $R_i(A)$, $w_{R_i(A)}|R_i(A)| = w_{R_i(A)}|A|\rho_{i,A}$, is less than its benefit, $w_{R_j}|R_j| - w_{R_j}|R_j|\rho_{i,A} = w_{R_j}|R_j|(1-\rho_{i,A})$, where $w_{R_j}|R_j|$ and $w_{R_j}|R_j|\rho_{i,A}$ are the size of R_j before and after the semijoin, respectively.

2.4 Join Reducers and Gainful Semijoins

The application of join operations as reducers may result in more profitable semijoins available. Those semijoins which become profitable due to the use of join reducers are termed *gainful semijoins* [6]. Consider the query graph shown in Figure 2 with its profile in Table 1, for example. It can be verified that the semijoin $R_3 \xrightarrow{A} R_1$ is not profitable since $w_{R_3(A)}|R_3(A)| > w_{R_1}(1-\rho_{3,A})|R_1|$. Note that although this semijoin is not profitable, it is *gainful* if we perform $R_1 \Rightarrow R_2$ and $R_2' \Rightarrow R_3$ after this semijoin operation, where $R_1 \Rightarrow R_2$ means that we ship R_1 to the site where R_2 resides and join R_1

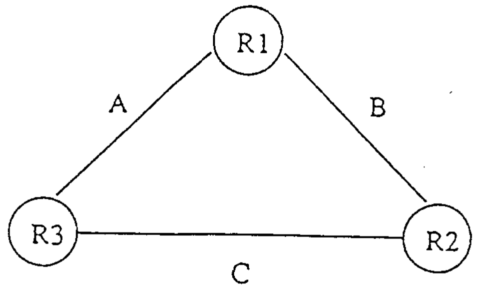

Figure 2: A query graph G_{EX2}

with R_2. It can be shown that for the total communication costs required, $|R_3(A)| + 2|R_1|\rho_{3,A} + 3|R_1 join R_2|\rho_{3,A}$ $\approx 2190 < 2|R_1| + 3|R_1 join R_2| = 2542$, meaning that it is advantageous, as far as the cost of data transmission is concerned, to perform $R_3 \xrightarrow{A} R_1$, $R_1' \Rightarrow R_2$ and then $R_2' \Rightarrow R_3$, instead of performing directly $R_1 \Rightarrow R_2$ and $R_2' \Rightarrow R_3$. Thus, it can be seen that whether a semijoin is gainful or not depends on the subsequent join operations. (Note that the effect of a join operation can be determined according to [7].) For simplicity, both the *profitable semijoins* and the *gainful semijoins* are called *beneficial semijoins*. In [6], they have proposed a polynomial time algorithm to find a sequence of join reducers. In [8], based on the *cumulative benefit* of a semijoin, they have used a heuristic to determine the set of beneficial semijoins to be interleaved into a given join reducer sequence. In [15], based on the *dynamic cumulative benefit* of a semijoin, they have applied a variant of the A^* algorithm to determine the set of beneficial semijoins to be interleaved into a given join reducer sequence.

2.5 Final and Non-final Joins

After identifying target relations in a tree query, we define those relations which are intermediate nodes in the paths between any two target relations as *related relations* [4, 5]. The union of target relations and related relations are called *final relations*. A relation which is not a final relation is called a *non-final relation*. For a join, if both of its joining relations are final relations, we call it a *final join*. Otherwise, we call it a *non-final join* [4]. For the join tree shown in Figure 3, if the target list contains $R_1.A$ and $R_4.C$, then R_1 and R_4 are target relations, R_2 is a related relation, and the final relations are R_1, R_2, and R_4. The final joins are $R_1 \xleftrightarrow{A} R_2$ and $R_2 \xleftrightarrow{C} R_4$.

3 The Algorithm

Given a tree query, we can construct a query graph $G = (V_Q, E_Q)$. According to final joins and non-final joins, we can divide this query graph into two parts: 1) the *final query tree*, and 2) the *non-final query trees*. A *final query tree* contains those final relations and final joins only. A *non-final query tree* consisting of non-final joins and those relations which participate in these non-final joins, is a rooted

Relation R_i	$\|R_i\|$	Size of relation	Attribute X	$\|R_i(X)\|$	Selectivity $\rho_{i,x}$	w_X
R_1	620	1240	A	400	0.80	1
			B	600	0.60	1
R_2	700	1400	B	580	0.58	1
			C	450	0.75	1
R_3	778	1556	A	360	0.72	1
			C	480	0.80	1

Table 1: Profile for query G_{EX2} where $|A| = 500$, $|B| = 1000$, and $|C| = 600$

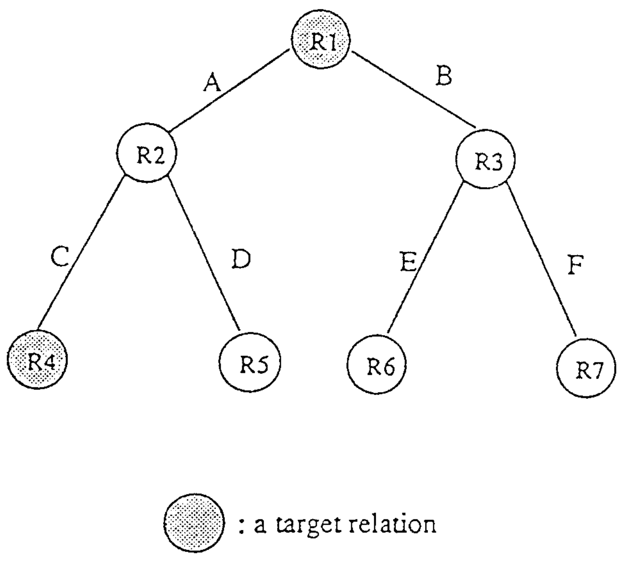

Figure 3: A query graph G_{EX3}

tree with its root node a final relation. A tree query can contain only one final query tree and zero or more non-final query trees. For example, given a query graph G_{EX3}, as shown in Figure 3, Figure 4-(a) shows the related final query tree and Figure 4-(b) shows the related non-final query trees, where we assume that R_1 and R_4 are target relations. Actually, to derive the final and non-final query trees, we simply identify the target relations first. That is why we call this approach a *target-relation-based* approach.

After finishing the above query graph analysis, we can first focus on those non-final query trees. For each of those non-final query trees, what we want to do is to fully reduce the root node at a cost as low as possible, which is a single reducer program problem. After fully reducing the root of each non-final query tree by using a single reducer program, we have reduced the size of each of the final relations. Therefore, we can reduce the data transmission cost for the final query tree. Moreover, when there is more than one non-final query trees, we can process them in parallel, which can shorten the query response time. Consequently, our target-relation-based approach not only can reduce the data transmission cost but also the response time. Moreover, the larger the number of non-final query trees is, the more reduction our approach can achieve.

Consider the following query:
select $R_2.A$, $R_4.C$
from R_1, R_2, R_3, R_4, R_5
where $R_1.A = R_2.A$ and $R_1.B = R_3.B$ and $R_3.C = R_4.C$ and $R_3.D = R_5.D$
with its query graph G_{EX4} shown in Figure 5

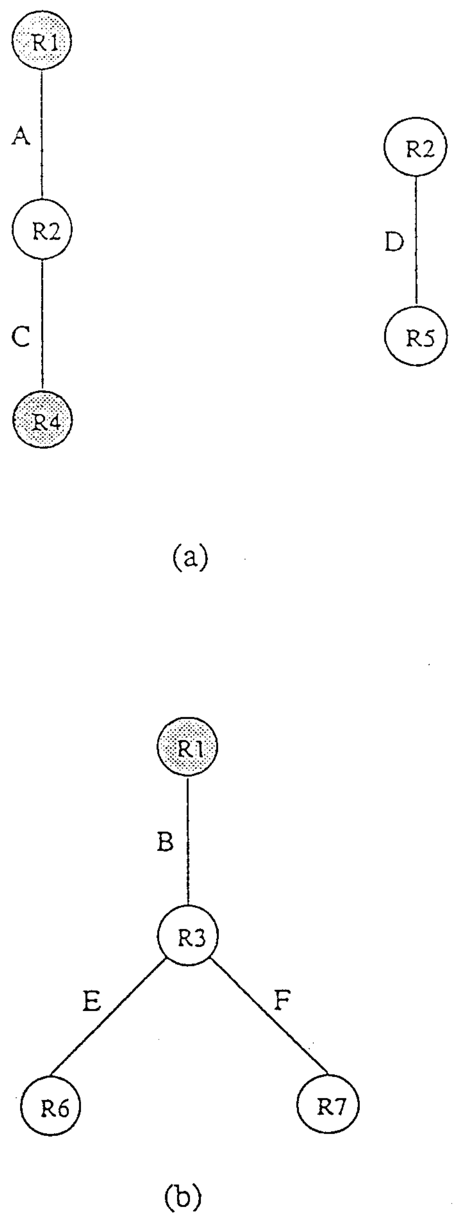

Figure 4: Query graph analysis for G_{EX3}: (a) a final query tree; (b) non-final query trees.

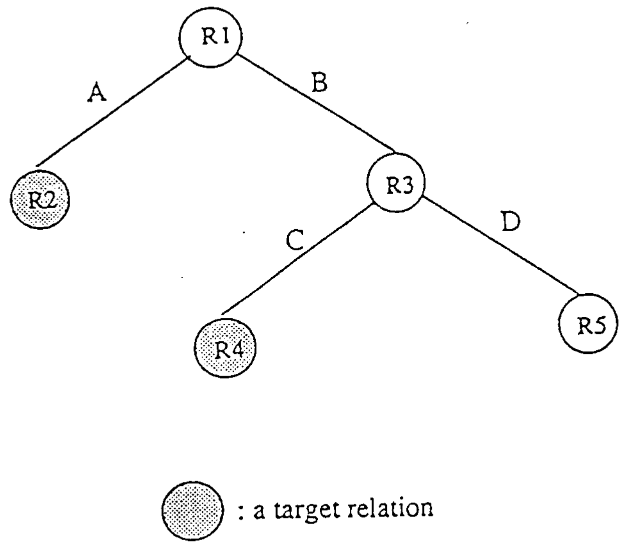

Figure 5: A query graph G_{EX4}

and data profile shown in Table 2. Suppose R_2 and R_4 are target relations and R_3 is the final site. Let's compare the data transmission cost to evaluate this query by the following three strategies.

Case 1: Using semijoins only as reducers for query processing [2].

For the data shown in Table 2, we observe that $R_3 \rightarrow R_4$ and $R_3 \rightarrow R_1$ are profitable semijoins and they should be executed first. The cost required for this two semijoins are 680 and 864. Next, we need $2112 + 3720 + 4692 + 3600 = 14124$ units of transmission cost to send relations R_1, R_2, R_4, and R_5 to the final site R_3. Therefore, the total transmission cost is $680 + 864 + 14124 = 15668$.

Case 2: Using joins and semijoins as reducers for query processing [8].

Instead of sending all relations toward the final site, we would like to use joins as reducers and perform $R_1 \Rightarrow R_2$ and then $R_2' \Rightarrow R_3$. In all, the transmission cost for each step is as follows: $R_3 \rightarrow R_4$ (680), $R_3 \rightarrow R_1$ (864), $R_1' \Rightarrow R_2$ (2112), $R_2' \Rightarrow R_3$ (2908), $R_4' \Rightarrow R_3$ (4692), and $R_5 \Rightarrow R_3$ (3600). Thus, the total transmission cost in Case 2 is 14856, which is less than 15668 that is required in Case 1.

Case 3: Using the concept of target relations for query processing.

According to the query graph shown in Figure 5, we can obtain a final query tree shown in Figure 6-(a) and a non-final query tree shown in Figure 6-(b), where R_2 and R_4 are target relations. For the non-final query tree, we would like to perform $R_5 \rightarrow R_3$ which needs $1040 \times 2 = 2080$ units of transmission cost. Therefore, the size of relation R_3 which should participate in the final join is reduced. Next, for the final query tree, we apply semijoins and joins together [8]; we perform $R_3 \rightarrow R_4$ (680), $R_3 \rightarrow R_1$ (864), $R_1' \Rightarrow R_2$ (2112), $R_2' \Rightarrow R_3$ (2908), $R_4' \Rightarrow R_3$ (4692). Then, the total transmission cost in Case 3

Relation R_i	$\lvert R_i \rvert$	Size of relation	Attribute X	$\lvert R_i(X) \rvert$	Selectivity	w_X
R_1	1100	3300	A	984	0.82	2
			B	1000	0.74	1
R_2	1240	3720	A	900	0.75	2
			E	800	0.80	1
R_3	1300	5200	B	864	0.64	1
			C	680	0.68	1
			D	1280	0.80	2
R_4	2300	6900	C	900	0.90	1
			F	920	0.92	2
R_5	1800	3600	D	1040	0.65	2

Table 2: Profile for query graph G_{EX4}, where $|A| = 1200$, $|B| = 1350$, $|C| = 1000$, $|D| = 1600$, $|E| = 1000$, and $|F| = 1000$

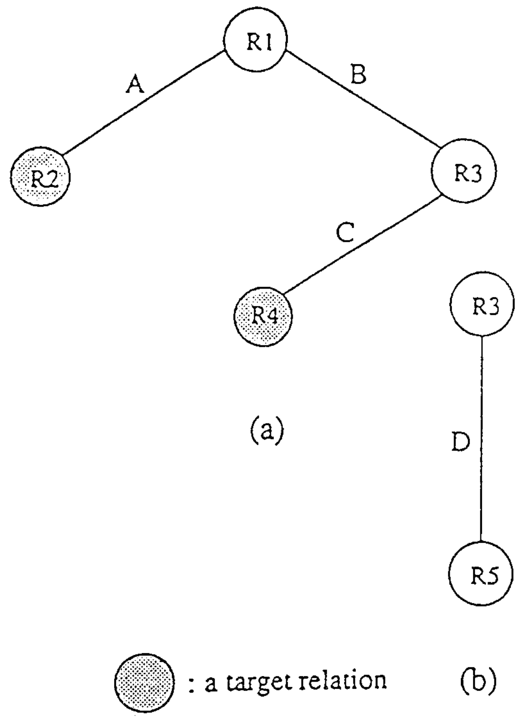

Figure 6: Analysis of query graph G_{EX4}: (a) a final query tree; (b) a non-final query tree.

is $2080 + 680 + 864 + 2112 + 2908 + 4692 = 13336$, which is less than 14856 that is required in Case 2.

4 Conclusion

In this paper, we have shown that by considering target relations, we can reduce more transmission cost than simply using semijoins only or combining semijoins and joins together. Moreover, the larger the number of non-final query trees is, the more reduction in transmission cost and response time our proposed approach can achieve.

References

[1] Peter M. G. Apers, Alan R. Hevner and S. Bing Yao, "Optimization Algorithms for Distributed Queries," *IEEE Trans. on Software Eng.*, Vol. SE-9, No. 1, pp. 57-68, Jan. 1983.

[2] Philip A. Bernstein and Dah-Ming W. Chiu, "Using Semi-Joins to Solve Relational Queries," *Journal of the Association for Computing Machinery*, Vol. 28, No. 1, pp. 25-40, Jan. 1981.

[3] Philip A. Bernstein, Nathan Goodman, Eugene Wong, Christopher L. Reeve and James B. Rothnie, "Query Processing in a System for Distributed Databases (SDD-1)," *ACM Trans. on Database Syst.*, Vol. 6, No. 4, pp. 602-625, Dec. 1981.

[4] Arbee L. P. Chen and Victor O. K. Li, "Improvement Algorithm for Semijoin Query Processing Programs in Distributed Database Systems," *IEEE Trans. on Computer*, Vol. C-33. No. 11, pp. 959-967, Nov. 1984.

[5] Arbee L. P. Chen and Victor O. K. Li, "An Optimal Algorithm for Processing Distributed Star Queries," *IEEE Trans. on Software Eng.*, Vol. SE-11, No. 10, pp. 1097-1107, Oct. 1985.

[6] Ming-Syan Chen and Philip S. Yu, "Using Join Operations as Reducers in Distributed Query Processing," *Proc. of the 2nd Int. Symp. Databases in Parallel Distributed Systems*, pp. 116-123, 1990.

[7] Ming-Syan Chen and Philip S. Yu, "Using Combination of Join and Semijoin Operations for Distributed Query Processing," *Proc. of the Int. Conf. on Distributed Computing Systems*, pp. 328-335, 1990.

[8] Ming-Syan Chen and Philip S. Yu, "Interleaving a Join Sequence with Semijoins in Distributed Query Processing,"*IEEE Transactions on Parallel and Distributed Syst.*, Vol. 3, No. 5, pp. 661-621, Sept. 1992.

[9] Ming-Syan Chen and Philip S. Yu, "Combining Join and Semi-Join Operations for Distributed Query Processing," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 3, pp. 534-542, June 1993.

[10] D. -M. Chiu and Y. -C. Ho, "A Methodology for Interpreting Tree Queries into Optimal Semijoin Expressions," *Proc. of 1980 ACM-SIGMOD Int. Conf. on Management of Data*, pp. 169-178, 1980.

[11] D. -M. Chiu, P. A. Bernstein and Y. -C. Ho, "Optimizing Chain Queries in a Distributed Database System," *SIAM J. Comput.*, Vol. 13, pp. 116-134, Feb. 1984.

[12] A. Hevner, "Query Optimization in Distributed Database Systems," Ph.D. Dissertation, Univ. Minnesita, 1979.

[13] Alan R. Hevner and S. Bing Yao, "Query Processing in Distributed Database Systems," *IEEE Trans. on Software Eng.*, Vol. SE-5, No. 3, pp. 177-187, May 1979.

[14] Ye-In Chang and Bor-Miin Liu, "An Eulerian-Path-Like-Bassed Algorithm for Semijoins," *Proc. of the 1994 International Computer Symposium*, Vol. II, pp. 1043-1047, Dec. 1994.

[15] Ye-In Chang and Bor-Miin Liu, "A Heuristic Algorithm for Beneficial Semijoins," *accepted by the 8th International Conference on Parallel and Distributed Systems*, Sept. 1995.

[16] Sakti Pramanik and David Ittner, "Optimizing Join Queries in Distributed Databases," *IEEE Trans. on Software Eng.*, Vol. 14, No. 9,pp. 1319-1326, September 1988.

[17] Hyuck Yoo and Stephane Lafortune, "An Intelligent Search Method for Query Optimization by Semijoins," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 1, No. 2, pp. 226-237, June 1989.

[18] C. T. Yu and C. C. Chang, "Distributed Query Processing," *ACM Computing Surveys*, Vol. 16, No. 4, pp. 388-433, Dec. 1984.