A cyclic multi-relation semijoin operation for query optimization in distributed databases

S. Bandyopadhyay+, Q. Fu
School of Computer Science
University of Windsor
Windsor, Ontario N9E 4L7, Canada
email: subir@cs.uwindsor.ca

and

A. Sengupta
Department of Computer Science
University of South Carolina
Columbia, SC 29208, USA
email: gupta@cs.sc.edu

Abstract

In this paper we have introduced the concept of multi-relation semijoin and we have shown that it is possible to get substantial reduction in the cost of data communication if a restricted form of this operation called cyclic multi-relation semijoin is used for query optimization in distributed databases. We have proposed a heuristic to identify situations where the cyclic multi-relation semijoin operation is likely to be useful. To study the applicability of this operation and to determine how much additional benefit we may expect, we have augmented a well known semijoin based algorithm with this operation. We have used simulation studies with a large number of queries and our experiments indicate that depending on the characteristics of the database, improvements ranging from 30% to 80% is possible.

1. Introduction

There has been considerable research in the area of query optimization in distributed databases particularly in the areas of semijoin strategies [1-8]. It has been established that finding an optimal semijoin strategy is, in general, an NP-complete problem[9] leading to heuristics to obtain a near optimal solution. The semijoin is often not an efficient filter since many tuples in one or more of the relation may actually not be needed in computing the result but the semijoin operations fails to eliminate all unneeded tuples. A join query graph is convenient [7] to represent the joins that have to be carried out and is defined as **G = (V, E)** where each node in **V** represents a relation and each edge in **E** corresponds to an equi-join predicate. Consider as an example the following.

Example 1: select B, D, F, G, H, I **from** R_1, R_2, R_3, R_4, R_5 **where** R_1. A = R_5. A **and** R_1. B = R_2. B **and** R_2. C = R_5. C **and** R_2. F = R_3. F **and** R_3. E = R_4. E **and** R_4. D = R_5. D

0-7803-5979-8/00 $10.00 © 2000 IEEE

The schemas of R_1, R_2, R_3, R_4 and R_5 are {A, B, G}, {B, C, F, H}, {E, F}, {D, E} and {A, C, D, I} respectively. The corresponding query graph and the database profile are shown in Fig 1 and tables 1 and 2. Here |X| denotes the cardinality of set X, $\Pi_X R_i$ denotes the projection of relation R_i on attributes in set X and $\omega_{\{X\}}$ the sum of sizes (in bytes) of all attributes in set X. We use the term *edgelabels*(**G**) to denote the set of all labels of attributes in the query graph **G**. For example, if **G** is the query graph shown in Fig 1, then *edgelabels*(**G**) = {A, B, C, D, E, F}.

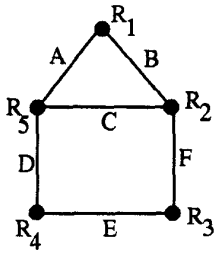

Fig 1: A query

Table 1:

Relation R_i	$\|R_i\|$	Attribute X	$\|\Pi_X R_i\|$
R_1	1190	A, B	1000,1100
R_2	3440	B, C, F	900, 1000, 480
R_3	1180	E, F	800, 585
R_4	3100	D, E	780, 800
R_5	2152	A, C, D	800, 1000, 600

Table 2:

Attribute X	|X|	$\omega_{\{X\}}$
A	1000	2
B	1200	1
C	1200	3
D	800	1
E	1000	1
F	600	1
G	1000	1
H	1000	1
I	1000	1

This is a slightly modified version of an example given in [7]. If a number of relations $R_1, R_2, \ldots, R_p$ are joined, the following theorem from [7] may be used to estimate the number of tuples in the resulting relation.

Theorem 1: Let $G = \{V, E\}$ be a join query graph and $S = \{V_S, E_S\}$ be a connected subquery graph of G. Let $R_1, R_2, \ldots, R_p$ be the relations corresponding to the nodes of V_S and let $edgelabels(G) = \{A_1, A_2, \ldots, A_q\}$. The expected number of tuples in $N_T(S)$ in the relation resulting from joining all the relations in S is given by the following where m_i is the total number of times A_i appears as an attribute in the relations in S and $|A_i|$ represents the number of distinct values of attribute A_i.

$$N_T(S) = \frac{\prod_{i=1}^{p} |R_i|}{\prod_{i=1}^{q} |A_i|^{(m_i - 1)}}$$

In this paper we propose a new database operation and we show by experiments that this operation, when used in conjunction with the semijoin operation is often a better filter than semijoin alone. As in most work, we use the SPJ type query [1] in this paper and we will investigate strategies based on semijoin operations to reduce the communication cost. We use Algorithm General [3] as a starting point in our studies.

Our aim in this paper is *not to* propose a definitive heuristic for query optimization but to test the viability of the new operation within an existing algorithm. We show that it is possible to significantly improve semijoin-based heuristics for optimization by using an operation we call the **cyclic multi-relation semijoin** operation.

The join of R_i and R_j at the site of relation R_k will be denoted by $R_i \overset{k}{\bowtie} R_j$. We will use $\overset{j}{\blacktriangleright\!\!\blacktriangleleft} R_i, P$, (where R_i is a relation that appears in predicate P) to denote the join of all R_i satisfying P at the site of relation R_j. For example, for the join query graph shown in Fig 1, $\overset{2}{\blacktriangleright\!\!\blacktriangleleft} R_i, R_i \varepsilon \{R_1, R_2, R_5\}$ will reperesent the operation $R_1 \overset{k}{\bowtie} R_2 \overset{k}{\bowtie} R_5$. In section 2, we develop the notion of a *multi-relation semijoin operation*. In section 3, we extend it to the concept of *cyclic multi-relation semijoin operation* and describe a heuristic for identifying situations where this operation is likely to reduce communication costs. In section 4, we describe how we incorporate this operation into a well known optimization heuristic and studied our heuristic on a large number of sample queries.

2. Multi-relation semijoin

Given a connected subquery graph $S = (V_S, E_S)$ of query graph $G = (V, E)$ and a node $R_i \varepsilon V_S$, the multi-relation semijoin operation $R_i \ltimes V_S$ is a composite database operation which reduces the number of tuples in R_i by a sequence of database operations involving relations in V_S at the site of relation R_i. This is a semijoin operation since it does not involve communication of all attributes of relations in V_S and is defined as $R_i \ltimes V_S =$

$$R_i \overset{i}{\bowtie} \overset{i}{\blacktriangleright\!\!\blacktriangleleft} \prod_X R_j, (R_j \in V_s - R_i)$$

Here $X = schema(R_j) \cap edgelabels(S)$ is the set of attributes of R_j that appears as *edgelabels* in S. This multi-relation semijoin operation may be interpreted as

- Project R_j on attributes that appears at least twice in relations in V_S. This is done for all relations in V_S except R_i (since R_i cannot reduce itself),
- Communicate all these projected relations to the site of R_i and
- Carry out a multi-operand join of R_i with all these projected relations.

Example 2:

For the graph shown in Fig 1, we consider the subquery graph $S = (V_S, E_S)$ where $V_S = \{R_1, R_2, R_5\}$, and E_S is the set of the edges connecting nodes in V_S. We carry out the operation $R_2 \ltimes V_S$ as follows:

Edgelabels(S) = {A, B, C} so that
$schema(R_1) \cap edgelabels(S)$ =
$\{A, B, G\} \cap \{A, B, C\}$ = {A, B} and
$schema(R_5) \cap edgelabels(S)$ =
$\{A, C, D\} \cap \{A, B, C\}$ = {A, C}. Thus
$\prod_{schema(R_1) \cap edgelabels(S)} R_1 = \prod_{A,B} R_1$ and
$\prod_{schema(R_5) \cap edgelabels(S)} R_1 = \prod_{A,C} R_1$ so

that $R_2 \ltimes V_S = R_2 \overset{2}{\bowtie} \Pi_{A,B} R_1 \overset{2}{\bowtie} \Pi_{A,C} R_5$. We use the term multi-relation since our query graph, in general, refers to a number of relations. After the operation, the schema of the relation stored at i is

The cost of the operation $R_i \ltimes V_S$ is the total amount of data that has to be communicated from sites of all relations R_j, $R_j \in V_S - \{R_i\}$ to the site of relation R_i.

Since $\left|\Pi_{schema(R_j) \cap edgelabels(S)} R_j\right| \leq |R_j|$ this cost cannot exceed

$$\sum_{j \in S - \{i\}} \omega_{(schema(R_j) \cap edgelabels(S))} \times |R_j|$$

.

If the size of the relation R_i after the operation is less than the original size of the relation, then the operation is beneficial. This benefit may be estimated by using theorem 1as

The *net benefit of the operation* $R_2 \ltimes V_S$ is the difference between the benefit and the cost. For instance, in example 2, where no prior database operation has taken place, we may estimate the number of tuples from the operation $R_2 \ltimes V_S$ using theorem 1. Projection, in general, reduces the number of tuples; however, in the absence of information, we have no way to estimate how much reduction will take place after the two projections $\Pi_{A,B} R_1$ and $\Pi_{A,C} R_5$. We can conservatively assume that the size of $\Pi_{A,B} R_1$ and $\Pi_{A,C} R_5$ is the same as the size of R_1 and R_5. so that the estimated number of tuples after the operation is $(|R_1| \times |R_2| \times |R_5|)/(|A| \times |B| \times |C|)$ = $((1190 \times 3440 \times 2152)/(1000 \times 1200 \times 1200))$ = 6. The cost to send $\Pi_{A,B} R_1$ to the site of R_2 is (2+1) * 1190 = 3570 and the cost to send $\Pi_{A,C} R_5$ to the site of relation R_2 is (2 + 3) * 2152 = 10760. Thus the total cost for this operation is 14330. This dramatic reduction in the expected number of tuples in the result is a consequence of applying theorem 1. So long as the distribution of common attribute values are random, any set of instances of relations R_1, R_2 and R_5 will have the same expected number of tuples. Our experiments indicate that our calculations are quite accurate.

It is easy to verify that in this example, no semijoin on R_1, R_2 or R_5 is profitable. The cost to send the original R_2 to the query site is $3440 \times 6 = 20640$. The total cost to carry out the multi-relation semijoin operation and sending the result to the query site is $14330 + 6 \times 6 = 14366$.

In this example, there are other important spin-off advantages of this operation since the selectivities of all attributes in this result are very low so that the number of tuples in the result is 6. For instance, the selectivity of attribute F in the result is 6/600 = 0.01. We may use these attributes in the result as very effective reducers for other relations. In order to use this operation effectively, we need to identify subgraphs $S = (V_S, E_S)$ and a site containing say relation R_i such that the net benefit of the operation $R_i \ltimes V_S$ is likely to be high.

Property 1[1]: If $S = (V_S, E_S)$ is a subquery graph and there exists a node $R_v \in V_S$ such that all edges in E_S incident on that node R_v are labelled with the same attribute X then the net benefit of $R_i \ltimes V_S$ where $i \neq v$ cannot exceed the net benefit of the following sequence of database operations:

a) $R_i \rtimes R_v$ for some relation R_j containing attribute X.

b) $R_i \ltimes V_S - \{R_v\}$

Property 2: If $S = (V_S, E_S)$ is a subquery graph and there exists a node $R_v \in V_S$ such that all edges in E_S incident on that node R_v are labelled with the same attribute X then the net benefit of $R_v \ltimes V_S$ cannot exceed the net benefit of the sequence of database operations.

a) $R_j \ltimes V_S - \{R_v\}$ where R_j contains attribute X

b) $R_v \rtimes R_j$

Properties 1 and 2 allow us to concentrate only on

1. Proof of these properties are omitted .

those subquery graphs where a) each relation has two or more edges in the subquery graph and b) there are at least two distinct attributes in the labels of edges from any given relation. Such subquery graphs have the following property:

Property 3 : Each node in a subquery graph as described above lies on 1 or more cycles.

3. Cyclic multi-relation semijoin

The number of cycles in a graph may be very large and determining all cycles in a graph is an exponential problem[11]; finding overlapping cycles is intractable. We need a simple and computationally feasible method to identify candidate subquery graphs and to determine where the multi-relation semijoin operation should be carried out.We now present a heuristic to address these two problems.

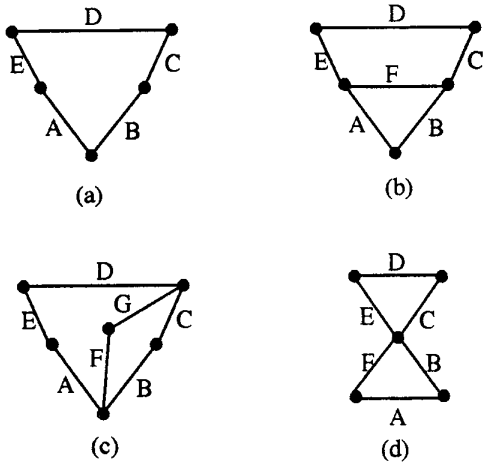

Fig 2: Some promising subquery graphs

This heuristic has two parts and is easy to implement but it does not allow us to identify all candidate subquery graphs:

a) **Identify a candidate subquery graph**: Our heuristic is to identify cycles such that the successive edges in each cycle have distinct edges.

b) **Identify the location for the semijoin operation**:If R_j is one of the nodes in a subquery graph and the site of the multi-relation semijoin is distinct from the site of R_j, the cost to send relevant attributes to the query site is . Once a cycle has been identified using part a of our heuristic, the site with maximum data transmission cost, is our location of the operation.

Part a of the heuristic will not be able to identify situations shown in Fig 2c or Fig 2d since they involve more than one nonoverlapping cycles. It does identify situations of overlapping cycles where the nodes in one cycle is a subset of the nodes in another cycle. This situation occurs in Fig 2b. Part b of the heuristic takes into consideration only the cost of the operation and ignores the fact that, if R_i and R_j are two relations on a cycle identified using part a of our heuristic, the benefit of the multi-relation semi-join carried out at the site of relation R_i will be different from the benefit of carrying out the operation at R_j.

Due to the fact that we restrict our consideration only to cycles in the query graph we use the term *cyclic multi-relation semijoin* (CMS) operation and we will use $R_i \ltimes V_S$ to denote this operation on a subquery graph $S = (V_S, E_S)$, carried out at the site of relation R_i where S is a cycle where no two successive edges in the cycle have the same label. We now discuss our heuristic for identifying candidate subquery graphs for any given query graph $G = (V, E)$. There are three stages in this process:

Stage i) We identify cycles in the query graph where no two successive edges have the same label.

Stage ii) We check each cycle to find embedded cycles. Each cycle is augmented by 0 or more edges from the query graph giving us a promising subquery graph S where all nodes satisfy properties 2 and 3.

Stage iii) We find a relation $R_i \ltimes V_S$ such that the CMS operation $R_i \ltimes V_S$ may be carried out at the least communication cost.

In stage i, we use a heuristic **H** to construct trees similar to spanning trees with the condition that for any interior node v , V in the tree, the label of the edge from the parent(v) to v is different from the label of the edge from v to any of its children. During this process we carry out a breadth first search and check for cycles that pass through the root node. Our heuristic guarantees that we do not generate duplicate cycles. The steps of the heuristic **H**and an analysis of its complexity are omitted due to lack of space. In stage ii, to find embedded cycles, we start with a cycle **C** of the query graph **G = (V, E)** identified by **H** to identify a candidate subquery graph $S = (V_S, E_S)$. V_S is the set of all nodes in the cycle **C**. We include in the set of edges E_S all edges (v, w) ε **E** such that v, w ε V_S. In this procedure, we include not only all edges in the cycle **C** identified by **H** but also other cycles **C1** such that the set of nodes in **C1** is a subset of the set of nodes in cycle **C**. This procedure is simple and details are omitted. The complexity of stage ii, for each cycle identified in stage i, is $O(|V| \cdot |A|)$ where A = edgelabels(G).

Example 3:
We apply our three stage procedure for finding overlapping cycles in the query graph shown in Fig 1

.

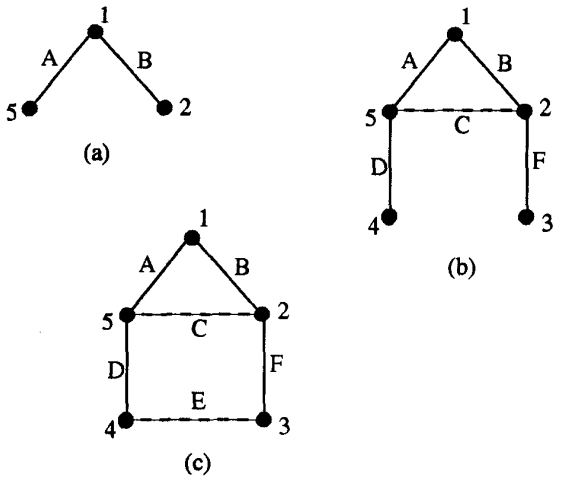

Fig 3: Determining cycles

We show the process using Fig 3 for h = 1. In Fig 3a the edges of 1 have been processed. Next we process the edges from 2 and 5, excluding the edges to their parent node 1. We also exclude any edge from 2(5) that has the same label as B(A). Fig 3b shows the situation when an edge is from node 2 to node 5 (shown with a dashed line since this will not be included in the tree) which is already in the tree. The edge from 2 to 5 has a label C which is distinct from the label A of the edge from 5 to 1. Also the first common ancestor of 2 and 5 is the root node 1. We conclude that this edge from 2 to 5, together with the paths from 5 to the root and from 2 to the root is a cycle. This cycle (2 - 5 - 1 - 2) has distinct labels in its successive edges and passes through the root. We therefore include this cycle in our list of cycles. This edge from node 2 to 5 in graph G is marked so that it will not be used again to generate the same cycle twice. Proceeding in the same way, as shown in Fig 3c, the edge from 3 to 4 gives us another cycle (1 - 2 - 3 - 4 - 5 - 1). We now delete node 1 and all associated edges from our graph and proceed with node 2. In a similar way, we will get a third cycle (2 - 3 - 4 - 5 - 2). We have to carry out stage ii for each of the cycles. We show this only for the second cycle (1 - 2 - 3 - 4 - 5 - 1) which illustrates how we get overlapping cycles.

We form a table (Table 3) with all join attributes as column headings and all nodes in the cycle as row headings. If the relation R_i contains an attribute in column j, we put a 1 in cell(i,j). The attributes with a column total of 2 or more are attributes that are permitted to appear in the edge labels of a promising subquery graph. In this case, our set of overlapping cycles corresponds to the query graph given in Fig 1 and consists of two cycles (1 - 2 - 3 - 4 - 5 - 1) and (1 - 2- 5- 1).

Table 3:

	A	B	C	D	E	F
1	1	1	0	0	0	0
2	0	1	1	0	0	1
3	0	0	0	0	1	1
4	0	0	0	1	1	0
5	1	0	1	1	0	0
Total	2	2	2	2	2	2

In stage iii we determine the site of operation. We show this only for the cycle (1 - 2 - 5 - 1). Here, the subquery graph $S = (V_S, E_S)$ has $V_S = \{1, 2, 5\}$ and $E_S = \{(1, 2), (2, 5), (5, 1)\}$). Here edgelabels(S) = {A, B, C}. We compute the cost of data communication from each node and are given by the following table.The fourth column in the table give the attributes in $schema(R_i) \cap edgelabels(S)$

Table 4:

Relation R_i	schema (R_i)	$\|R_i\|$	Common attributes	ω_X	$\|R_i\| \omega_X$
R_1	{A, B, G}	1190	{A,B}	3	3570
R_2	{B, C, F, H}	3440	{B, C}	4	13760
R_5	{A,C, D, I}	2152	{A,C}	5	10760

The site of operation is node 2 since the cost to send $\Pi_{B, C} R_2$ is the maximum.

4. Our query optimization heuristic and experimental results

As stated in the introduction, our motivation is to study whether the CMS operation is likely to be useful, in general. Using Algorithm General [3] as the starting point, we have added CMS including the 3 stage procedure given above. Since this heuristic is not the central theme of this paper and due to lack of space, we omit the details. We have processed a large number of queries to compare the relative efficiencies of Algorithm General and the modifications using the CMS operation. In our experiment, we have to generate a large number of queries and make a comparative study of the costs involved. We also need to study how the relative performances of the two approaches vary as the database profile changes. In this comparative study we actually carried out database operations on synthetic databases. This way we eliminated the uncertainties associated with the use of estimates in such performance analysis. Some important characteristics of queries and databases are a) the number of relations involved in the query, b) the degree to which relations are connected to each other, (measured by the ratio of the actual number of edges between different relations and the number of edges in a completely connected query graph), c) the range of selectivities of different attributes and d) the number of tuples in the relations. In order to study different types of queries, we have considered the following 12 types of queries described in table 5. The domain size for all the queries is between 500 and 600. The simulator(written by. Bealor [10]) takes a query of a given type and generates a) a random database profile, b) a random query graph satisfying our specifications with respect to the characteristics given above and c) synthetic databases on which to carry out database operations. These databases were populated with tuples satisfying the database profile generated by the simulator.

Table 5:

Query type	Number of relations	Number of attributes	Connectivity	Selectivity range	Relation size
1	3	1 - 3	60 - 70%	0.9 - 1.0	1500 - 2000
2	3	1 - 3	60 - 70%	0.6 - 1.0	1500 - 2000
3	3	1 - 3	60 - 70%	0.9 - 1.0	5000 - 6000
4	3	1 - 3	60 - 70%	0.6 - 1.0	5000 - 6000
5	6	1 - 4	40 - 50%	0.9 - 1.0	1500 - 2000
6	6	1 - 4	40 - 50%	0.6 - 1.0	1500 - 2000
7	6	1 - 4	40 - 50%	0.9 - 1.0	5000 - 6000
8	6	1 - 4	40 - 50%	0.6 - 1.0	5000 - 6000
9	6	1 - 4	40 - 50%	0.9 - 1.0	1500 - 2000
10	6	1 - 4	40 - 50%	0.6 - 1.0	1500 - 2000
11	6	1 - 4	40 - 50%	0.9 - 1.0	5000 - 6000
12	6	1 - 4	40 - 50%	0.6 - 1.0	5000 - 6000

We now process the same query twice - once using Algorithm General and then using our heuristic using CMS, giving us two schedules of database operations and two estimates of communication costs for the two approaches. Then we carry out, on the synthetic database, the schedule of database operations generated by each of the two methods, giving 2 actual communication costs. In our study we repeated this process for fifty queries for each of the 12 query categories. We show the results of our experiments in table 6 below: Each row gives the total communication costs for all 50 queries in a given category. We have also

included the communication costs using no optimization where we simply ship all relations participating in a query to the query site.

Table 6:

Query type	Communication cost using		
	No optimization	AHY	Our heuristic
1	15615	15615	4892
2	15646	12351	3791
3	49588	47574	17567
4	49267	33696	13121
5	30024	30024	5782
6	30356	20078	4089
7	95186	88105	33720
8	94423	46790	22107
9	35555	35555	4099
10	35831	19656	3116
11	111719	94771	21452
12	109348	36570	10384

5. Conclusions

In this paper, we have introduced the notion of a multi-relation semijoin operation and have shown that it is a useful operation. This operation shares the characteristic of join operations that, after the operation, the schema of the result contains new attributes. Like the semijoin operations, this operation does not involve the transmission of attributes which are not involved in the database operation. In many cases, the reduction in communication cost is dramatic. We found that this operation works particularly well in situations where the result of the query has relatively few tuples but the selectivities of each semijoin reducer is 1 or close to 1. We also found that the first multi-relation semijoin is crucial. If this operation gives a dramatic reduction in the number of tuples, other significant reductions in the number of tuples in other relation are likely to follow. Our use of formulae 2a-2c is also important in our approach. Our simulation experiments show that the improvements over the semijoin operations are considerable. We would like to point out that, there is no single best strategy for query optimization and depending on the database profile and the query graph, there are situations where the join, the semijoin, the bloom join or the multi-relation semijoin may be the best strategy. We conclude that optimization heuristics could be improved by taking into account all the characteristics of both the query as well the database profile and applying the most suitable technique

Acknowledgment: We used the implementation of Algorithm General, the query generator and the simulator for distributed queries developed by Mr. W. T. Bealor[10]. To help this study, Mr. Bealor also modified his original implementation to include our modifications with respect to the multi-relation semijoin.

References

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

[2] M. T. Ozsu and P. Valduriez, *Principles of distributed database systems*, Prentice Hall, 1991.

[3] P. M. G. Apers, A. R. Hevner and S. B. Yao, "Optimization algorithms for distributed queries", *IEEE Trans. Software Engineering*, SE-9, No 1, pp 57-68, 1983.

[4] L. Chen and V. Li, "Improvement algorithms for semijoin query processing programs in distributed database systems", *IEEE Trans. on Computers*, TC-33, No 1, pp 959-967, 1984.

[5] J. K. Mullin, "Optimal semijoins for distributed database systems", *IEEE Trans. On Software Engineering*, Vol 16, No 5, pp 558-560, 1990.

[6] C. Wang, V. O. K. Li, and A. L. P. Chen, "Distributed query optimization by one-shot fixed precision semijoin execution", *Proc 7th Inter. Conf. on Data Engineering*, pp 756-763, 1991.

[7] 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, Sept 1992.

[8] M. S. Chen and P. S. Yu, "Combining join and semijoin operations for distributed query processing", *IEEE Trans. Knowledge and Data Engineering*, Vol 5, No 3, 1993.

[9] A. R. Hevner, "The optimization of query processing on distributed database systems", Ph. D. Dissertation, Purdue University, 1979.

[10] W. T. Bealor, "Semi-join strategies for total cost minimization in distributed query processing", M. Sc thesis, University of Windsor, 1994.

[11] F. Harary, *Graph Theory*, Addison Wesley, Reading, MA, 1969.