2002

An evaluation of improvement algorithms for query processing in a distributed database system.

Nelson Chung Ngok. Chu
University of Windsor

An Evaluation of Improvement Algorithms for Query Processing in A Distributed Database System

By
Nelson Chung Ngok CHU

A Thesis
Submitted to the Faculty of Graduate Studies and Research
Through the School of Computer Science
In Partial Fulfillment of the Requirements for
The Degree of Master of Science at the
University of Windsor

Windsor, Ontario
Canada
2002

0-612-75823-0

972714

Abstract

In distributed database query processing, the database management system (DBMS) may consider all alternative queries and choose the one with the least cost. However, the number of alternative queries follows the number of relations, attributes, and distributed sites to increase exponentially. Therefore, the problem of finding an optimal query is a well-recognized NP-hard problem [CL84, OV99]. Applying heuristic algorithms to this kind of problem is a commonly used strategy. There are two basic steps in query processing. First, enumerate alternative plans for evaluating a query. Second, estimate the cost of each enumerated plan and choose the plan with the least estimated cost from the result of cost evaluation. In the area of distributed querying processing, the semijoin is a well-recognized operator, which provides efficient query results. There are some heuristic algorithms proposed to solve query-processing problems in distributed database systems. Unfortunately, most of these algorithms do not guarantee the optimality of the result. Therefore, some researchers have been motivated to identify some optimality properties for semijoin programs and have proposed a set of algorithms to improve a non-optimal semijoin program for satisfying those optimality properties. The performance and limitations of this set of algorithms will be evaluated in this thesis. There are some modifications on the algorithms which can improve the performance of improvement algorithms in this thesis. The research work includes the study of modification of the essential operations, such as semijoin, and the interrelationship between the sub-procedures in the algorithms. Different implementation approaches to the algorithms also have been explored.

Keywords: Distributed, Database, Join, Optimization, Optimal Query, Semijoin, Strategy

To my family

Acknowledgements

I consider it a great accomplishment to have completed this M.Sc. thesis under the direction of my advisor, Dr. Morrissey. After working under her for a period of time, I found that she is so knowledgeable in the field. I was very fortunate to obtain advice from her during my graduate studies; it was truly a good experience to receive her support and encouragement. Also, I gratefully acknowledge my committee members, Dr. Diana Kao, Dr. Scott Goodwin, and Dr. Akshai Aggarwal, for their individual contributions and comments to this thesis. I would like to thank all supporting staff in my department.

The successful completion of this thesis has been helped immensely by the love and support of family and friends. Thank you my parents and brothers for being so supportive, kind, and loving.

Table of Contents

List of Figures

Chapter 1. Introduction

Distributed query processing is a process to transform a high-level query language of distributed databases to a low-level database language for retrieving the database in an efficient and effective strategy. Query processing is required in distributed database system (DDBS) due to the dynamic conditions in distributed system and the requirements changing in business environment. Distributed data processing provides cost-effective solution to satisfy the adaptive situation and also considered some business issues: cost, scalability, integration of different software modules, integration of legacy systems, new applications, and market forces [KOS00]. A good design of a query processing system is including many difficulties as mentioned above. The system needs to handle all data/query communications in a distributed computer network. In recent ten years, the size of many networks becomes huge and each site in the network may have different networking configuration. As mentioned before, the adaptive distributed environment becomes one of the most challenging issues.

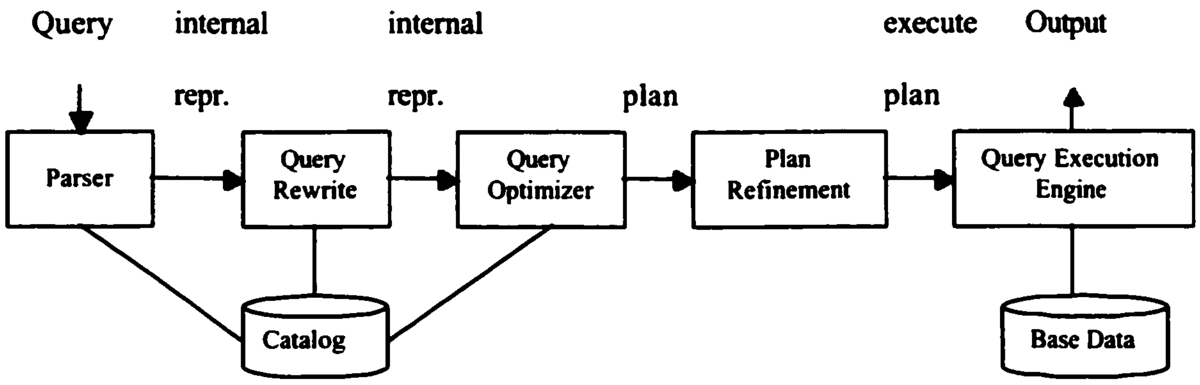


Figure 1.1 Phases of query processing [KOS00]

A general query processing architecture has been shown in figure 1.1. This architecture can be applied in any database system, such as parallel, centralized, and distributed. The query processor, parser, receives a query then translates the high-level query language into low-level database language. In the next step, the query will be optimized and generate a query execution plan. The plan passes to the query execution engine to execute and produce the query result [KOS00]. The other commonly accepted distributed query processing architecture includes three phases: local processing, reduction, and final query processing. Local process in first phase includes the selection from the relation and projection on the joining and target attributes. In the reduction phase, semijoin is commonly used to eliminate the relation size and to reduce the communication costs. In the last phase, final query processing, the query site will collect and reassembly the reduced relation for producing the result of query [YC84, KR87].

The other important issue, the processing evaluation/cost estimation, should be considered in the distributed query processing architecture. In order to determinate the performances of DDBS, there are three measurable criteria: local processing cost, communication cost, and total cost. Local processing costs refer to disk access time, CPU processing time. In a network, the communication is one of the main sources of the cost. The total cost measures the sum of the costs of transferring data [YC84]. Beside the cost-base evaluation, we also can use some other aspects to evaluate a DDBS: such as overhead, accuracy, and complexity. Overhead is expressed in terms of delay and the number of message throughout during the strategy execution. Accuracy of query strategy means the strategy whether or not to correct or eliminate the non-beneficial operations. Complexity refers to how much computational resources have been consumed [BRJ89].

In order to provide a cost effective solution for the query processing, there are many different techniques, semijoin is one of the most popular used in distributed environment, and algorithms have been developed that based on different criteria assumption.

However, finding an optimal query is commonly recognized as an NP-hard problem [CL84, OV99] in distributed query processing. The number of query follows the number of relations, attributes, and distributed site to increase exponentially. Therefore, finding an optimal query plan is a significant challenge. Applying heuristic algorithms to solve this kind of problem is a commonly used strategy. [CL84] identified four optimality properties of a semijoin query program and a set of algorithms to improve a non-optimal semijoin program. There are some limitations behind the algorithms and also some implementation issues did not discuss in the paper. Therefore, a sequence of research works will be extended form [CL84] in this thesis.

The rest of this thesis will be divided in five chapters. The literature review on query processing strategy planning and evaluation, query processing techniques/operators, and query processing optimization algorithms are described in chapter 2. Chapter 3 introduces the improvement algorithm for semijoin query program. The improved algorithm will be illustrated in chapter 4. The evaluation of the algorithms will be discussed in chapter 5. Chapter 6 will concludes this thesis work and discusses the future works.

Chapter 2. Literature Review

2.1. Distributed Database System

A Distributed Database System (DDBS) combines two computer technologies, database system and computer network, to provide an information management mechanism. A formal definition from [OV99] is "a distributed database is a collection of multiple, logically interrelated databases distributed over a computer network". A DDBS supports the information management in an organization and minimizes the geographic limitation on information exchange between offices or subsidiaries within the organization. It should provide stable and reliable data communication environment for data exchanging in transparent system [KOS00].

There are many lower level implementation details of database system and computer network in a DDBS. In a transparent system, the lower level implementations should be hidden from users. Users do not need to consider these implementations details when using or developing their applications. In different DDBS, the combinations of different types of transparencies and other criteria should be handled in the development processes. The followings are the most important transparencies that should be provided by a DDBS [OV99, SAL96].

Data independence: It refers to logical data independence and physical data independence. Logical data independence means the external schema is shielded from any modification of conceptual schema of a relation. Physical data dependence refers to the conceptual schema being shielded from changes to the physical schema or physical implementation of relation or database.

Distribution transparency: It does not require users to know the location of data and the topology of the database system. It means there is no difference between the data or database application located, for example, in centralized and distributed database.

Replication transparency: Users need not to handle how and where to get and/or modify data in a single or multiple sites. DBMS takes the responsibility of these jobs.

Fragmentation transparency: It is similar to replication technique for performance, reliability, and availability of the database system. It is not the responsibility of a user to know which fragmentation technique has been applied in the system. The data retrieval method will be taken care of by the database application and the DBMS.

Reliability: It refers to the correctness and consistency of the results of requests provided by the DBMS. It relates to the access coordination, communication control, and query processing strategies. There are some techniques to increase reliability: transaction, distributed reliability protocol, two-phase commit, and distributed recovery protocol.

Scalability: In a distributed environment, there are many dynamic elements in the system. Therefore, a DDBS should have ability to increase or decrease the number of sites and users. The dynamic change may also cause the movement of data from one database or site to another.

Distributed systems can be built in many networking topologies. There are two main categories of computer network, Local Area Network (LAN) and Wide Area Network (WAN). The communication cost exists in the network due to the computer nodes or database systems physically distributed in different locations. Therefore, the data/message takes time to travel across the network from the sender to receiver. Different combinations of query operations, algorithms, and network results in communication cost differences. There are many research related to the relationship between distributed database, query operations and algorithms, and network topologies. For example, [LC85] did some research work on the performance of distributed join algorithms in a LAN and [SAL96] proposed an economic paradigm as the solution to solve the problems in WAN.

The experiment in [LC85] concluded some characteristics in a LAN. First of all, communications costs are not the primary dominant factor that affects performance of the distributed database system in LAN. Secondly, the pipelined nested loop join provides better performance for a small size of relation while large relations benefit from pipelined sort-merge methods in local network. On the other hand, the higher selectivity of a join attribute introduces fewer data storage accesses, read or write. As a consequence, fewer data pages are accessed in indexed nested loops join operations and fewer accesses from remote sites. Therefore, the communication costs have been reduced.

In WAN, different sites may have different accessing algorithms, site-specific data type, and constraints on servicing remote requests. These factors introduce difficulties in scheduling distributed actions in a large system with the large number of combination of actions. The dynamic environment of each site increases the complexity of a distributed database system. The proposed economic model in [SAL96] reduces the scheduling complexity of distributed interaction without using global synchronization.

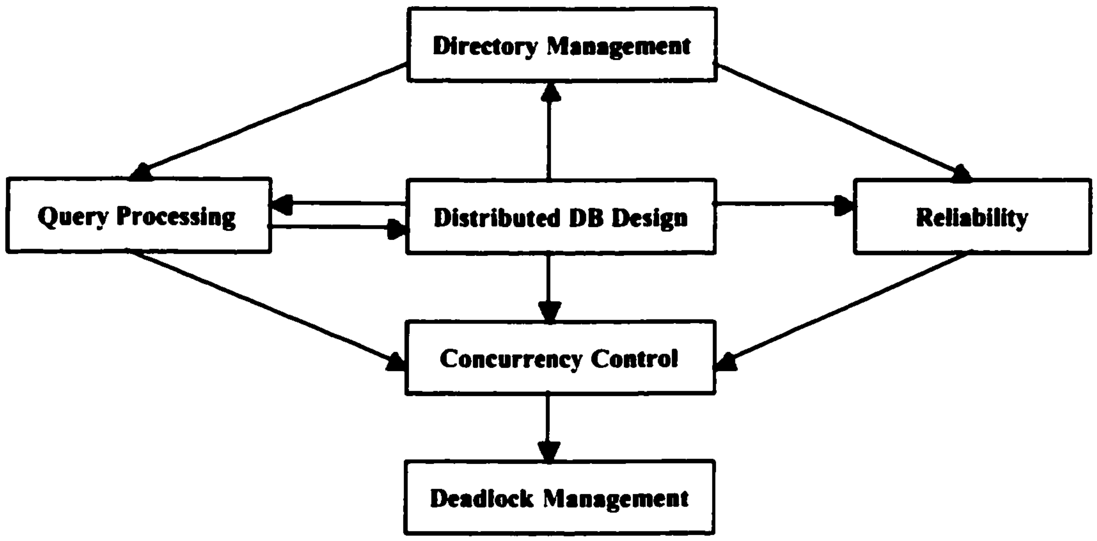


Figure 2.1 Relationships Among Research Issues [OV99]

In order to understand the problems associated with distributed database system. A DDBS can be divided into distributed DB design, directory management, concurrency control, deadlock management, query processing, and reliability. The relationships between components of a DDBS are shown in figure 2.1. The critical section is the distributed database design. It affects directory management, concurrency control, query processing, and reliability directly. The definition of fragmentation, placement policies, and management strategies of directory management and query processing strategies can be affected by the distributed database design. The decision of directory management also affects the query performance and system's reliability. In the same manner, concurrency control based on the different inputs, of DB design, query process, and reliability of system, to decide the concurrency control policies and strategies. The deadlock management definitely depends on the employment of concurrency control policies and strategies [OV99].

2.2. Query Processing Strategy Planning and Evaluation

2.2.1. The Strategy Planning During Query Processing

A good distributed query processing strategy should accommodate many DDBS criteria or transparencies in its distributed environment. It also should be able to provide an optimal plan(s) for executing the query. The basic requirements of a good strategy are that the query plans be effective, efficient, and as simple as possible to implement. In order to achieve these goals, the strategy should able to reduce the communication costs and redundancies, at least using redundant relations to reduce the communication cost. In addition, the strategy needs to support different data placement techniques: such as fragmentation, replication, and caching [KOS00]. Different query operators have its characteristics and optimization power. Therefore, the strategy should choose appropriated operator and technique for query processing that satisfy different requirements. Besides providing the ability to use the optimization techniques, there are many problems that need to be handled during the strategy planning process. In the rest of this section, an overview of these problems would be discussed.

The Copy Identification Problem

In a distributed database, the relations are possibly fragmented, replicated, or cached. It depends on the placement policies. If replication has been employed, then the selection of which copy of the relation to access is very important. The selection affects the cost of the query and maintenance of the relation/database. The query must ensure all replicated relations have been updated correctly. In some cases, there may exist a trade-off between query cost and maintainability. Algorithm Pre-Processing and Algorithm Opt-Site are proposed in [YC83]. Algorithm Pre-Processing used to eliminate the dominating sites, relations, and selects the essential relation. It should be executed before the Opt-Site which is used to find a primary copy and a number of secondary copies of each unfragmented relation that is referenced by the given query. The set of primary copies of the relations are contained in the minimum number of sites. If relation, R_i, is to be reduced, then the primary copy of R_i should be used. No secondary copy of the relation is used to reduce other relations. If R_i has not been reduced, a secondary copy of R_i may be used to reduce another relation.

Elimination of Unnecessary Relations

In general, many query algorithms consist of two processing phases: semijoin process and join process. Semijoin operations are used to reduce the relations' size and join operations are used to join the result from pervious phase one for answering the query. Not all results from phase one are beneficial to the join operations in phase two. Transmission of unnecessary relation(s) to phase two will increase the communication cost. Therefore, these relations should be eliminated.

Replacement of Semi-joins and Identification of Useless Semijoins

Some semijoins are redundant or can be replaced by another optimal semijoin(s) and provide better result, reduction of the cost. In some cases, the rearrangement of execution order of the semijoin helps to improve the performance [BGW81] and the useless semijoins should be discarded. The rearrangement techniques of semijoin operation and reassembling site would be discussed in chapter 3.

Transmission of Complement

The cost of a query refers to the amount of data needed to be transferred. If the amount of the possible attribute values of a query is more than half of that attribute, then transmitting the complement of the possible values should be considered. It reduces the cost of transmission. Of course, the query strategy needs to handle the complement data. [YC83] indicates the improvement from twenty percent to a few hundred percent for simple queries of two or three relations but the improvement depends on the selectivity of the relations.

2.2.2. Dynamic Query Strategies

A single query plan cannot cover the entire possible range of dynamic query constraints, data distributions, and resource situations in distributed environment. If the plan used for an extended period of time, re-optimization of the plan is required. In traditional query optimization, the compiled optimization application/plan is activated by procedure call and the plan reflects the situation of database at compile time only. This approach reduces the optimization overhead during execution and provides higher transaction rates. In consequence, it provides faster query evaluation. Unfortunately, it also is a major drawback because the query strategies cannot reflect the dynamic constraints of distributed environment during executing period. There are two main categories of dynamic constraints: program variable and system load. Program variable includes cardinality of the relation and selectivity of selection predicate. System load includes many system resources: such as buffer pages, and number of processor. The situation of the database may be changed during the time between compilation and execution. The plan cannot accurately estimate the optimal plan for the query at execution time. Therefore, the dynamic query evaluation and re-optimization system is very important for a distributed database system.

The simplest method to build an effective query strategy is including all query execution plans together as a complete strategy. It is an inefficient method and not possible for large amounts of complex queries, even simple queries. Therefore, [GW89] has suggested two simpler methods. First, a subset of all query execution plans can be selected. Second, we can store the elements of the plans instead of the complete plans. When the plan is called then link the elements together.

2.2.3. Dynamic Query Evaluation Techniques

Each query strategy execution should proceed through three phases: monitoring phase, decision making phase, and corrective phase [BRJ89]. In monitoring phase, a processor monitors the progress of the strategy execution. Decision making phase, decides which non-optimal strategies produced in the formulation process need to be corrected because of inaccurate estimation. Finally, the current strategies will be discarded and the corrected strategies will be applied during the corrective phase.

Monitoring

The query strategies are monitored to guarantee consistency, accuracy, and correctness during the execution. The monitoring processor may also involve monitoring the program variable and system load [GW89] mentioned in pervious section. There are three monitoring methods suggested in [BRJ89]. First, distributed monitoring, any partial information, such as cost, size, and overhead, of strategies execution will be broadcasted to all processors. It helps to decide the non-optimal strategy. In centralized monitoring, a master processor collects the progress information of the strategies execution from other processors and decides which strategy needs to be correct. Individual monitoring means each processor uses local-information to decide whether the strategies are optimal or not. The information will be sent to neither master nor all processors.

Decision Making

The incorrect strategies were determined during monitoring phase. The actual cost is far from the estimated cost due to the inaccurate estimation used in the formulation of the strategies. There are two techniques, reformulation and thresholds, described in [BRJ89] for improving these problems. In reformulation process, the un-executed portion of the query strategies will be reformulated to produce new strategies which have less cost than the original strategies. The thresholds technique formulates and sets up the upper and lower limits for each parameter. If the actual value of the parameter falls between upper and lower limits, then the strategy is incorrect. There are some issues needed to consider during the development of the correction policies. First, the correction processes, including reformulation and threshold checking, introduce overhead cost. Second, estimated value is always different from actual value. Therefore, the monitoring and decision-making process needs to reserve some space for this fact. Finally, the difference between actual value and estimate value does not mean that strategies are non-optimal. It may because the strategy already is the best in all strategies.

Corrective Action

Similar to monitoring phase, there are three correction methods: distributed, centralized, and individual [BRJ89]. In distributed correction, all processors will pass around their own result and finalize a final strategy to be executed. The master of centralized correction method will release the new corrective strategies to all other processor to execute. In individual correction scheme, the individual processor will abort the old strategies and broadcast the new strategies to all other processors. Synchronization is required for the distributed and individual corrective action because the corrected strategies are affecting the whole distributed database. Therefore, the updating processor needs to be synchronized with others to make sure only one is updated at a time.

2.3. Query Processing Techniques

2.3.1. Join

Join operation is one of the most useful operations in relational algebra and is the most commonly used way to combine information from two or more relations. Although a join can be defined as a cross-product followed by selections and projections. Furthermore, the result of a cross-product is typically much larger than the result of a join. The most general version of the join operation, R $\bowtie$c S, accepts a join condition, c, and a pair of relation instances, R and S, as arguments, and returns a relation instance, R'. The join condition is identical to a cartesian product of the pair of relation, R x S, followed by selection with the same condition, δc. The operation is denoted as follows [RG00]:

$$R \bowtie c S = \delta c(R \times S) = R'$$

Example: $R \bowtie_{(A>C)} S \rightarrow R'$

R	
A	B
1	2
3	4
0	null

S	
C	D
2	5
4	6

R'			
A	B	C	D
3	4	2	5

Figure 2.2 Join Operator

Figure 2.2 shows a join operation between relations, R and S, with a join condition, A > C. In this operation, the values of attribute A, in R, and attribute C, in S, will be compared. All tuples that fulfill the joining condition then forms a new relation, R'. In distributed database environment, join operation can reduce the local processing cost. Moreover, join operation minimizes the overhead message during the processes. A general procedure is described in [LPP91] to optimize the join queries in distributed environment with non-fragmented relations. The general procedure is divided in to four main steps.

a) Sequencing optimization: the best sequence of binary joins is selected to execute n-ary joins.

b) Optimal materialization: for each relation to be retrieved, which may exist in multiple copies in the system or may be fragmented horizontally or vertically the optimal site for materializing it is chosen.

c) Distribution: the optimal allocation of binary join executions as well as the storing of intermediate result relations amongst the available sites is determined.

d) Execution strategy: a binary join can be implemented by means of a number of techniques. at different costs; for instance it can be implemented as a nested loop join or as a merge scan join.

During the optimization process, we need to consider which site(s) will take place to execute the join operation(s) and also all execution sequences. The decisions of both problems affect the processing cost. An algorithm is proposed in [LLP91] which helps to optimize the join operations in distributed environment. It constructs an Adorned binary Query Tree (AQT) structured solution and helps to reduce the search space. An AQT is a Binary Query Tree (BQT) with labels to specify the execution site and storing site of join operations. A BQT is a Query Tree (QT) that has two nodes only for each non-leaf node. In a QT, the leaves represent the relations to be joined and each leaf has a list of relation's site that can be used for the materialization. Each node represents a join operator and it will send the result back to the corresponding root to prepare the final result that sending back to the query site.

2.3.2. Semijoin

Semijoin is one of the most popular operators used in distributed database environment for combining information of two relations over the set of attributes. It can decrease the number of tuples that need to be handled to form the join. As a results, it can reduce the data transmission between sites for the query. A semijoin operation, $R \ltimes_P S$, take two relation instances, R and S, as arguments over a predicate, P, and returns a relation instance, R'. Semijoin is denoted as follows:

$$R \ltimes_P S = R'$$

Example: $R \rtimes_{R.P\# = S.P\#} S \rightarrow R'$ [MOR01]

R	
P#	Name
2	Adams
3	Smith
5	Kelly

S	
P#	Status
1	250
2	300
4	450
6	275

R'	
P#	Status
2	300

Figure 2.3 Semijoin Operator

In figure 2.3, it takes two relations R and S, and does the semijoin operation with the predicate, R.P# = S.P#. There are four steps for a semijoin operation. First, project R over the joining attribute P#. Second, ship R[P#] to the site of S, and ship the relation R to query site when the query require some information not exist in S. Third, execute R[P#] $\bowtie$ S. Finally, ship the result, R', from the site of S to query site.

In a distributed database system, semijoins have to be performed in relation-to-relation or a relation-to-fragment manner to avoid eliminating contributive tuples. Semijoin cannot apply to two fragments because it may eliminate some tuples before the comparisons with all tuples of the operations [CL90].

2.3.3. 2-Way Semijoin

2-way Semijoin is an extended version of the semijoin, more cost-effective operator to reduce relation size. 2-way Semijoin can be seen as two semijoins, such as $R_i \leftarrow A \rightarrow R_j = \{ R_i [A] \rightarrow R_j,\ R_j' [A] \rightarrow R_i \}$, during the process because they have some functionality and result. It enhances the semijoin with backward reduction capability for more cost-effective query processing.

Example: $R_i \leftarrow A \rightarrow R_j$

R_i [A]

A
1
2
3
4
5
6

R_j

A	B
2	2
4	4
5	6
6	8
8	10
10	12

R_j'

A	B
2	2
4	4
5	6
6	8

R_i [A] matching = (2,4,5,6)
R_i [A] not matching = (1,3) → used for backward reduction

Figure 2.4 2-Way Semijoin

Figure 2.4 shows the general procedures of 2-way semijoin. First, send the projection R_i[A] from site i to j. Second, reduce R_j by eliminating tuples whose attribute A are not matching any of R_i[A] and result in R_j'. During reduction of Rj, partition R_i[A] into R_i[A]m and R_i[A]nm where R_i[A]m is the set of values in Ri[A] which match one of R_j[A] and R_i[A]nm is R_i[A] - R_i[A]m. Third, send either R_i[A]m or R_i[A]nm whichever is less in size from site j back to I. Finally, reduce R_i using either R_i[A]m or R_i[A]nm. If R_i[A]m is used, then tuples whose attribute A are not matching any of R_i[A]m are eliminated. If R_i[A]nm is used, then tuples whose attribute A are matching one of R_i[A]nm are eliminated.

The proofs in [KR87] shows the 2-way semijoin having more reduction power and propagation of reduction effects than semijoin by applying backward reduction technique. The data sending back is smaller and always guaranteed to remove tuples in R_i. Even if the number of "not matching" is 0; then the cost and benefit both are 0. It still is a cost effective method. The simulation in [KR87] also has demonstrated the possibility of replacing or combining two operators can provide better result for a query.

2.3.4. Pipeline N-way Join

A pipeline N-way join algorithm is based on a 2-way reduction strategy, described in pervious section, combining the cache technique. The main idea of this algorithm is reducing the I/O cost (time) and the transmission cost across the network. It helps query site to minimize the frequency of disk access, read and/or write, during the N-way join processes. In the process, there is no intermediate results. Tuple connectors are used to replace the reduced relation and stored in main memory. Therefore, it does not require accessing the disk. The connectors can join together to produce a pipeline cache planner that used to synchronize the generation of query results.

There are three phases proposed in N-way pipeline algorithm: local processing phase and forward reduction, backward reduction and collecting phase, and pipeline execution phase.

Relation: R1		
R1	a	b
R11	4	b
R12	3	a
R13	1	a
R14	1	b
R15	0	a

Relation: R2		
R2	b	c
R21	a	x
R22	c	y
R23	b	w
R24	b	z
R25	a	y

Relation: R3		
R3	c	d
R31	z	m
R32	t	n
R33	y	n
R34	x	n
R35	y	k

Relation: R4		
R4	d	e
R41	l	1.2
R42	k	3.5
R43	n	6.4

Figure 2.5 Relations for N-way Pipeline Join

Connector: C1	
R1	b
R11	b
R12	a
R13	a
R14	b
R15	a

Connector: C2		
b	R2	c
a	R21	x
b	R23	w
b	R24	z
a	R25	y

Connector: C3		
c	R3	d
z	R31	m
y	R33	n
x	R34	n
y	R35	k

Connector: C4	
d	R4
k	R42
n	R43

Figure 2.6 Tuple Connectors

For example, a query executes a sequence of 2-way semijoin for four relations of figure 2.5. In the forward reduction and local processing phase, the projection of join attribute will be transferred for forward reduction and the tuple connector will be constructed at the end of this phase (see figure 2.6). The tuple connector is a projection of the relation on all the joining attributes and a tuple identifier (TID).

R1	R2	R3	R4
R12	R25	R35	R42
R13	R25	R35	R42
R15	R25	R35	R42
R12	R25	R33	R43
R13	R25	R33	R43
R15	R25	R33	R43
R12	R21	R34	R43
R13	R21	R34	R43
R15	R21	R34	R43

Figure 2.7 The Pipeline Cache Planner

Connector: C1	
R1	b
R12	a
R13	a
R15	a

Connector: C2		
B	R2	c
A	R21	x
A	R25	y

Connector: C3		
c	R3	d
y	R33	n
x	R34	n
y	R35	k

Connector: C4	
d	R4
k	R42
n	R43

Figure 2.8 Tuple Connectors Reduced by the Backward Reduction

In second phase, the tuple connectors are used for backward reduction and joined together to construct the pipeline cache planner (see figure 2.7). The planner is a N-ary relation with all joinable tuples of N relations and used to synchronize the generation of the result. Figure 2.8 shows how the tuple connectors are reduced during the backward reduction. The pipeline cache planner provides the information to reduce the size of the connectors of each relation. Figure 2.9 shows how the pipeline cache planner is constructed. It provides all information for required pieces of the query. In last phase, pipeline execution phase, the query site receives the pipeline cache planner and used to synchronize the assembly process of query result from N sites.

Built at Site 1			
R1	R2	R3	R4
R12	R25	R35	R42
R12	R25	R33	R43
R12	R21	R34	R43
R13	R25	R33	R43
R13	R25	R35	R42
R13	R21	R34	R43
R15	R25	R33	R43
R15	R25	R35	R42
R15	R21	R34	R43

Built at Site 2			
b	R2	R3	R4
a	R21	R34	R43
a	R25	R35	R42
a	R25	R33	R43

Built at Site 3		
c	R3	R4
x	R34	R43
y	R33	R43
y	R35	R42

Figure 2.9 The Construction of the Pipeline Cache Planner

2.3.5. Hash-semijoin

Hash-semijoin tends to provide more cost effective distributed query processing by using search filter and semijoin replacement techniques. It can minimize the transmission cost of a semijoin operation because the hash-semijoin operation transfers a search filter instead of the projection of a joining attribute of the relation. A search filter uses a bit array to represent the semijoin projection and a hash function to hash a bit address for each projection element. In some case, the search filter may accept a false drop due to the collision in hash function. The probability of false drop depends on the size of the bit array. The search filter is optimal when the bit array is half full.

The procedures of using a hash-semijoin described in [TC92]. First, we need to initialize a bit array of F bits to all be 0 and the size of the array is calculated by the equation, $F = (d / \ln 2) |R_i|$. Second, use the hash function to probe a bit address for each join attribute and set the corresponding bit to 1. Third, transmit the filter, bit array, to the requesting site. Finally, use the same hash function to hash the join attribute value to bit addresses. If the bit array's value is not correct, then discard the tuple.

The followings are the general steps for a backward replacement with Hash-Semijoin:

1. Remove the first element from the queue that used to record the nodes having no successors.
2. Calculate the potential cost of the traditional semijoin and hash-semijoin.
3. If hash-semijoin is beneficial to the query then replace semijoin with hash-semijoin.
4. Update the potential cost.
5. Insert the element into the queue according to its level.
6. Repeat the processes until the queue is empty.

2.3.6. Domain-Specific Semijoin

As mentioned before, semijoin is limited to be used in relation-to-relation or relation-to-fragment manner to avoid eliminating contributing elements. To solve this problem, Domain-Specific Semijoin is proposed in [CL90]. It can be performed individually without loss of tuples in fragment-to-fragment manner. It enhances distributed query processing to be more flexible. In domain-specific semijoin, R_{ik} (A = B] R_{jm}, A and B are joining attributes, and R_{ik} and R_{jm} are two fragments of joining relations R_i and R_j. The following is the definition of the domain-specific semijoin:

$$R_{ik}\ (A = B]\ R_{jm} = \{r \mid r \in R_{ik}\ ;\ r.A \in R_{jm}\ [B] \cup (\mathrm{Dom}[R_i\ .B] - \mathrm{Dom}[R_{jm}\ .B])\}$$

In a sequence of domain-fragment semijoins within a fragmented relation, the order of execution of domain-specific semijoins on the fragments is not affecting the result. The following steps are proposed in [CL90] for performing each domain-specific semijoin:

1. Calculate the estimate benefits and costs.

   Cardinality of the fragment after domain-specific semijoin:

$$|R'_{ik}| = R_{ik}(A = B]R_{jm} = |R_{ik}|\left(1-\frac{|Dom[R_{ik}.A]\cap Dom[R_{jm}.B]|}{|Dom[R_{ik}.A]|}\right)+|R_{ik}|\left(1-\frac{|Dom[R_{ik}.A]\cap Dom[R_{jm}.B]|}{|Dom[R_{ik}.A]|}\right)\left(\frac{|R_{jm}[B]|}{|Dom[R_{jm}.B]|}\right)$$

   The number of tuples reduced by domain-specific semijoin:

$$|R_{ik}|-|R'_{ik}|=|R_{ik}|\left(1-\frac{|Dom[R_{ik}.A]\cap Dom[R_{jm}.B]|}{|Dom[R_{ik}.A]|}\right)\left(1-\frac{|R_{jm}[B]|}{|Dom[R_{jm}.B]|}\right)$$

2. If it is found to be profitable, include it in the current query-processing strategy; otherwise, ignore it.
3. Update the related information in the database profile.

Example:

R1					|R11| = 2K;	|R12| = 1K	
x	S = Dom[R1.x]	|S|	w(R1.x)	i	|R1i[x]|	T = Dom[R1i.x]	|T|
A	{v | 1 ≤ v ≤ 2K}	2K	10	1	600	Same as R1.A	
				2	400	Same as R1.A	
C	{v | 1 ≤ v ≤ 10K}	10K	2	1	2K	{v | 1 ≤ v ≤ 4K}	4K
				2	1K	{v | 4K ≤ v ≤ 10K}	6K

R2					|R21| = 25K;	|R22| = 40K	
x	S = Dom[R1.x]	|S|	w(R2.x)	i	|R2i[x]|	T = Dom[R2i.x]	|T|
B	{v | 1 ≤ v ≤ 2K}	2K	10	1	300	{v | 1 ≤ v ≤ 1K}	1K
				2	700	{v | 1K ≤ v ≤ 2K}	1K
D	{v | 1 ≤ v ≤ 500}	500	5	1	300	Same as R2.D	
				2	400	Same as R2.D	
E	{v | 1 ≤ v ≤ 50K}	50K	6	1	25K	Same as R2.E	
				2	40K	Same as R2.E	

Figure 2.10 Database Profile for Relations R1 and R2 [CL90]

From the information of figure 2.10, the benefits and costs can be calculated by using pervious formulas.

$$\frac{|Dom[R11.A] \cap Dom[R21.B]|}{|Dom[R11.A]|} = 0.5 \qquad \frac{|R21[B]|}{|Dom[R21.B]|} = 0.3$$

|R'11| = |R11(A+B]R21| = (2K)(1-0.5) + (2K) x 0.5 x 0.3 =1.3K

R11(A=B]R21:

Benefit: Ctran(|R11| - |R'11|) x w(R11) = 0.7K x 12 = 8.4K

Cost: Ctran |R21| x w(R11) = 0.3K x 10 = 3K (profit = 5.4K)

R11(A=B]R22: Benefit = 3.6K Cost = 7K (not profitable)

R11<A=B]R2 (semijoin):

Benefit: 8.4K + 3.6K = 12K Cost: 3K + 7K = 10K (profit = 2K)

From above calculation, it shows R11(A=B]R21 providing more profit. Therefore, the domain-specific semijoin will replace semijoin. The last step is updating the database profile according to the estimation formulas.

The domain-specific semijoin is based on many assumptions. First, it assumes all values of each attribute, A, in fragment, R_{ik}, are randomly selected from Dom[R_{ik}.A]. Second, all tuples of R_{ik} are uniformly distributed over values of R_{ik}.A. Third, it assumes the values of each attribute are independent. Finally, the cardinality of each fragment, each attribute of each fragment, the domain of each attribute of each fragment, the domain of each attribute of each relation, and the width of each attribute of each relation should be available for the calculation.

2.3.7. Generalized Semi-join

The other problem of semijoin is not being able to handle a cyclic query alone. Cyclic query have cycles in their join graph and for which full reducers cannot be found. Therefore, the concept of "Generalized Semi-join" is proposed in [KYY82] to handle an arbitrary cyclic query. The basic idea of generalized semi-join is applying spanning tree technique to form a non-cyclic graph and use generalized semi-join to process the query. In [KYY82], it assumed X be an attribute set satisfying $(R_i \cap R_j) \subseteq X \subseteq R$ and defined generalized semijoin, $R_i \boxtimes R_j$, as following:

$$R_i \boxtimes R_j = R_i \bowtie R_j [X]$$

There are two basic steps for generalized semi-join. First, add new attributes with null value to each relation scheme in order to convert the query into a tree query. Second, apply generalized semi-join to the tree query. In the rest of this section, an example will been used to explain the procedures for a query processing using generalized semi-join (see figure 2.11). In the figure, every edge represents a semijoin between two relations and every circle represents a relation. Figure a shows a cyclic graph of a set of semijoin operations. First of all, we identify the spanning tree and the edges that involved in the cycle(s) from the cyclic graph (figure b). Edges F, G and H cause cycles in the graph. Second, replace the edges that not in spanning tree by a set of edges that goes along other edges from the source node to destination node (figure c). Edge F is replaced and merged by the edges between R2 and R3, R1 and R2. Edge G is replaced and merged by edges between R2 and R3, R1 and R2, R1 and R5. Edge H is replaced and merged by edges between R4 and R2, R1 and R2, R1 and R5, R5 and R6. The final spanning tree is shown in figure d. The last step is applying generalized semi-join to each edge on the corresponding attribute(s). The major drawback of generalized semi-join is the increment of communication cost for tree queries.

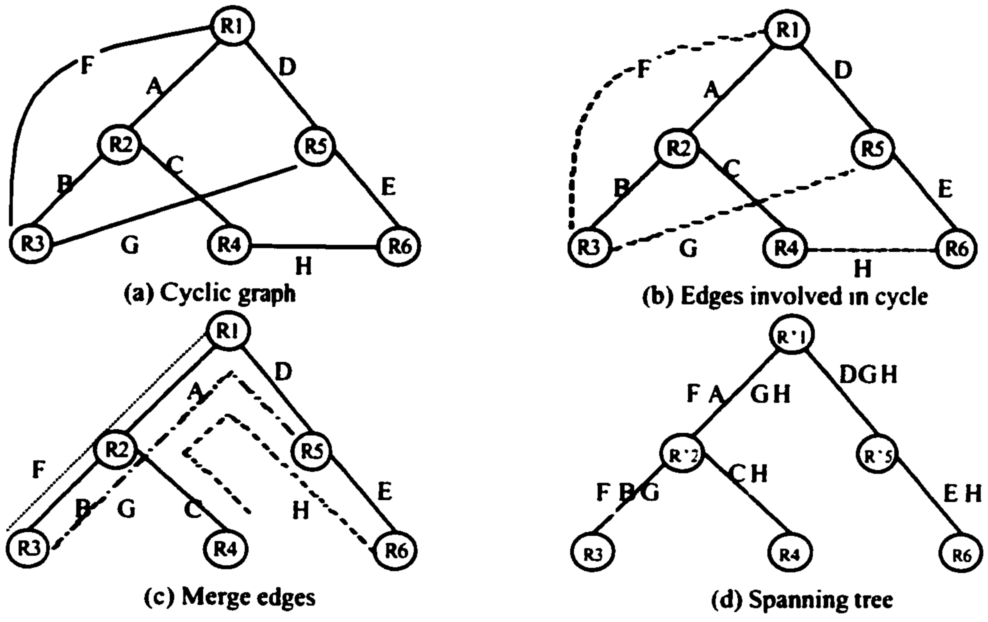


Figure 2.11 Generalized Semi-Join

2.3.8. Composite Semijoin

Many distributed query algorithms involving many operations, such as semijoins, have same source and destination sites. As a result, the cost, response time, will be increased and the performance becomes not efficient. It is because of retransmission of the same overhead for different operations with same source and destination sites. Composite semijoin is proposed in [PC90] to handle these problems by combining multiple semijoins to be a composite semijoin. A composite semijoin is a semijoin with a projection of multiple attributes involved in some operations with same source and destination sites. In some case, semijoin may not provide any benefit to query (see figure 2.12). There is no reduction when the single attribute semijoin is used, either sending D11 or D12 to relation 2. However, composite semijoin can reduce the cost by saving the repeated transmission of overheads and increasing the reducing power of the operation(s) (see figure 2.13). There is a significant reduction when a composite semijoin is used, sending composite attribute (D11 and D12) to relation 2.

Example [PC90]:
Semijoin

Relation 1	
D11	D12
1	a
1	b
2	c
3	c

$\ltimes$

Relation 2	
D21	D22
1	c
1	a
2	b
3	b

$\rightarrow$

Relation 2'	
D21	D22
1	c
1	a
2	b
3	b

Figure 2.12 Semijoin

Composite Semijoin

Relation 1	
D11	D12
1	a
1	b
2	c
3	c

$\ltimes$

Relation 2	
D21	D22
1	c
1	a
2	b
3	b

$\rightarrow$

Relation 2'	
D21	D22
1	a

Figure 2.13 Composite Semijoin

In [PC90], Composite semijoin has been applied in different algorithms, algorithm General (response time) [AHY83], algorithm W [PER85], and algorithm S [BGW+81]. The testing shows significant improvement from the original algorithms by using composite semijoin. It also points out algorithm General and some similar algorithms have ignored the reducing power of non-joining attributes. The strategy of composite semijoin is not guaranteed the reducing power, even the testing examples shows significant improvement in some case for composite semijoin. In some situations, composite semijoin may result in greater response time. Furthermore, this strategy is tested in static environment. Therefore, the dynamic constraints should be considered in development of the query processing with this technique. Combining traditional semijoin and composite semijoin is an option for improving both techniques.

2.3.9. Data Placement Strategies

Distributed database system, always involves multiple local databases. The distributed multi-database system provides full global database function with interactions between local databases. A distributed multi-database system has a global schema or global query system to query the distributed data. Data placement strategies refer to data allocation in distributed environment. One of the most concerned issues of data placement strategy development is the placement dependency. It is a data allocation constraint that can be used to increase the performance of query processing. Executing an update operation on a relation or fragment may violate global referential and placement dependency. In consequence, the correctness of query may be affected. Therefore, the violation must be handled [HCL98]. In order to handle these violations, we need to understand some placement techniques. There are two commonly used placement techniques, fragmentation and replication, will be described.

Fragmentation technique

Fragmentation breaks a relation into smaller relations or fragments, and stores at different sites. It increases the possibility of concurrent access the database system. There are two methods of fragmentation technique: horizontal fragmentation and vertical fragmentation. Horizontal fragmentation breaks relation into fragments that consists a subset of rows of the relation. The union of horizontal fragments must be equal to original relation (see figure 2.14). The fragments of vertical fragmentation consists a subset of columns of the relation. The collection of vertical fragmentation should be lossless decomposition (see figure 2.15). During the development of the data placement policies, there are some rules that must be enforced: completeness, reconstruction, and disjointness. Completeness refers to lossless decomposition property of normalization, no tuple loss in horizontal fragmentation and no attribute loss in vertical fragmentation. Reconstruction means the relations' constraints will be preserved during the fragmentation process. Disjointness ensures the horizontal fragments are disjoint and disjointness only defined on the non-primary key attributes of a fragment.

R		
A	B	C
1	5	6
2	7	8
3	9	2
4	3	4

=

F1		
A	B	C
1	5	6
3	9	2

+

F2		
A	B	C
2	7	8
4	3	4

Figure 2.14 Horizontal Fragmentation

R		
A	B	C
1	5	6
2	7	8
3	9	2
4	3	4

=

F1		
T_{id}	B	C
1	5	6
2	7	8
3	9	2
4	3	4

+

F2	
T_{id}	A
1	1
2	2
3	3
4	4

Figure 2.15 Vertical Fragmentation

Replication technique

Replication means a relation has more than one copy stored in one or more sites. It can increase the availability of data and supports faster query evaluation. Replication helps users to reduce the cost for accessing the data and also allows parallel processing of query. The major drawback is replication requiring huge overheads for updating and synchronizing the data in every site.

In distributed database system, updating operation can be divided in local update and global update. There are three kinds of updating operations: insertion, deletion, and modification. In local environment, insertion is an inserting operation of a new data into a relation or local fragment of a relation. Deletion or modification of an item is removing or modifying an item in a relation or a fragment of a relation. The replicated item in other site will not be removed or updated when the fragment or the relation is replicated in other site. In global environment, inserting a new item involves inserting the item into at least one fragment of relation in the whole distributed database system locally or inter-site. In the same manner, deletion and modification of an item involves deleting or updating all replicated items in all fragments of a relation in the database system [HCL98]. The placement policies and placement dependency definitely affect the updating processes and database management. Therefore, the above issues must be handled very carefully during the development processes of placement strategies.

2.4. Query Processing Optimization Algorithms

2.4.1. Optimization Algorithms for Distributed Queries

In order to optimize query processing in distributed environment, the methods commonly used are minimizing the response time and total time. Decomposing a query into sub-queries and executing in parallel help to reduce the response time and total time. The different situations in distributed environment may have different requirements of the performance, minimum response time or total time, of query strategies. Therefore, algorithm GENERAL is proposed in [AHY89] to deal with this problem. Algorithm GENERAL has three versions to handle different requirements: Response Time version, Total Time version, and Collective version. To minimize response time of a processing strategy, Algorithm GENERAL response time version emphasized the parallel data transmission. To minimize total time of a processing strategy, Algorithm GENERAL total time version emphasized the serial data transmission. To minimize data transmission redundancy, Algorithm GENERAL collective version produces strategies leading to a further reduction of the total time of a query.

There are four general steps in algorithm GENERAL. First, finish all local processing. Second, generate candidate relation schedules; Isolate each of the joining attributes, and define a simple query with an undefined result node. Algorithm PARALLEL helps to minimize response time for each simple query and Algorithm SERIAL helps to minimize total time for each simple query. This results in one schedule per simple query. Third, integrate the candidate schedules. For each relation, the candidate schedules are integrated to form a processing schedule. The integration is done by procedure RESPONSE, procedure TOTAL, or procedure COLLECTIVE. Procedure RESPONSE is used for response time minimization. Procedure TOTAL is used for total time minimization. Procedure COLLECTIVE also is used for minimizing total time and handling redundant data transmission. Finally, remove schedule redundancies. Eliminate relation schedules for relations which have been transmitted in the schedule of another relation. [AHY83]

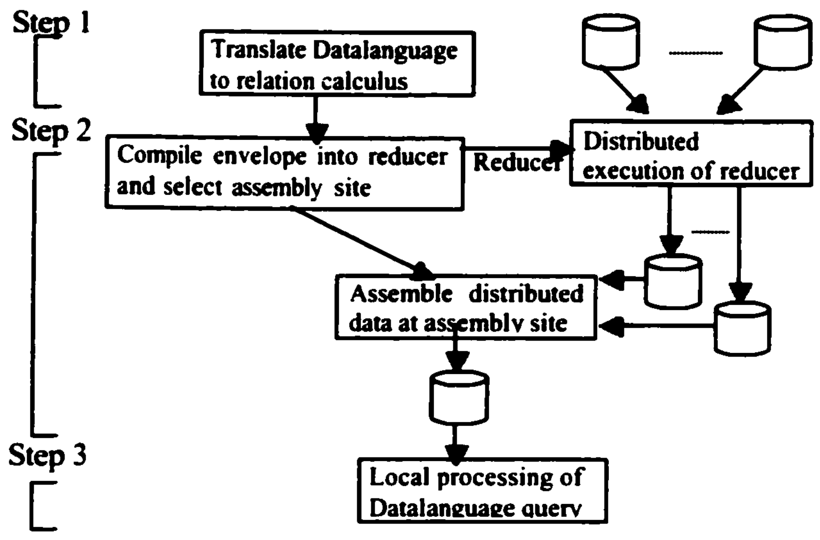
Procedure RESPONDE	Procedure TOTAL	Procedure COLLECTIVE
1. Candidate schedule ordering, in ascending order of arrival time 2. Schedule integration, construct an integrated schedule for the relation that consists of the parallel transmission. Select the schedule with minimum response time.	1. Adding candidate schedules 2. Select the best candidate schedule 3. Candidate schedule ordering 4. Schedule integration, construct an integrated schedule for the relation that consists of the parallel transmission. Select the schedule with minimum total time.	1. Select candidate schedule with minimum cost and selectivity < 1 2. Build processing strategy for parallel transmission 3. Test variations of strategy

In algorithm GENERAL, it has adopted algorithms, PARALLEL and SERIAL, which developed by Hevner and Yao. Algorithm PARALLEL can minimum response time schedules for a simple query, the relation contain only one attribute. It has employed in algorithm GENERAL (response time version) and responds to compute the minimum response time for each schedule. Algorithm SERIAL produce minimum total time schedule for a simple query. Algorithm GENERAL (total time and collective version) both use algorithm SERIAL to produce strategies to handle data transmission redundancy.

In algorithm PARALLEL, only parallel data transmission has been considered but the serial data transmission may also benefit to the query. Therefore, the combination of serial data transmission and parallel data transmission should be used to improve the performance. During construction of candidate schedule, algorithm PARALLEL suggests to have parallel transmission of the relation R_j and all schedules of relation R_k $(k < j)$. Some schedules may be not benefit to the query and it will increase the cost. Therefore, we should the best candidate schedule to perform the parallel transmission. In algorithm GENERAL, only forward reduction has been considered but the backward reduction technique also should be considered. We also should consider the possibility of combining other technique with semijoin, such as 2-way semijoin, hash-semijoin, and composite semijoin.

2.4.2. Response Time and Total Time Reduction

The idea of SDD-1 algorithm in [BGW+81] is employing semijoin to perform the reduction phase efficiently and minimize the transmission cost, data. The optimization idea of this algorithm is translating the query into a relational calculus form called an envelope. In this algorithm, there are three main steps (see figure 2.16). First step, maps a query, Datalanguage employed in SDD-1, into an envelope. An envelope is a relational calculus expression that maps a database into a sub-database. A good envelope tightly delimits the data needed by query. Second step, evaluates the envelope and translates it into reducer. A program contains relational operations and performs the reduction of the relation size. A reduction operation reduces the size of the database by eliminating data not specified by the envelope. It reduces the cost, inter-site data transfer, for the operation computation. [BGW+81] employs semijoin to implement the reduction operation. Third step, executes query at a site using the data assembled by second step. The following is graphical representation of above three steps:



Step 1
Step 2
Step 3
Translate Datalanguage to relation calculus
Compile envelope into reducer and select assembly site
Reducer
Distributed execution of reducer
Assemble distributed data at assembly site
Local processing of Datalanguage query

Figure 2.16 SDD-1 Query Processing Algorithm

Besides the basic algorithm, [BGW+81] proposed an optimization algorithm. This algorithm compiles envelop into a reducer, which is estimated to be profitable in any database modeled by database profile. It selects an assembling site and appends to reducer commands to move the reduced database to the site. This algorithm has three parts: initialization, main loop, and termination. In initialization section, reducer will be initialized to contain all local operations permitted by the envelope. In main loop, all non-local semijoins permitted by the envelope will be tested and selects the all profitable semijoin appending to reducer.

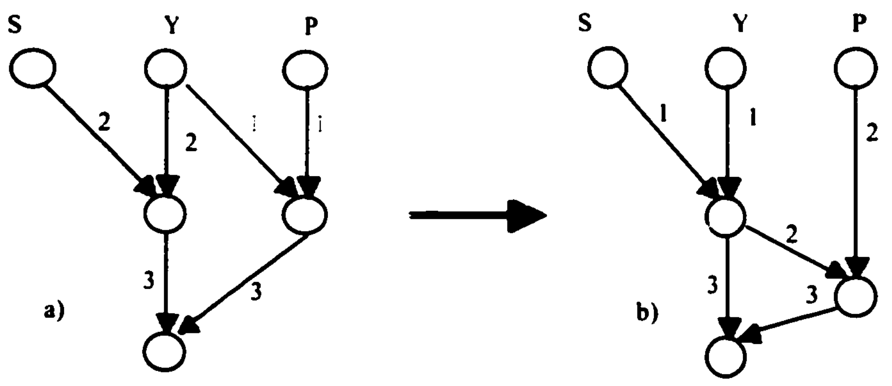


Figure 2.17 First Enhancement

There are two enhancements for the algorithm Opt has been mentioned in [BGW+81]. First enhancement permutes the order of semijoins in reducer to decrease the cost of some semijoins without increasing the cost of any others. In figure 2.17 a, semijoin 1 uses Y to reduce relation P, and semijoin 2 reduces Y. Since semijoin 2 reduces Y, the cost of semijoin can be decreased by delaying it until semijoin 2 executes (see figure 2.17 b). This permutation increases the effect of semijoin 1 and selectivity of semijoin 2. It also reduces the cost of subsequent semijoin. Second enhancement, prunes semijoins from reducer that are rendered unprofitable by the choice of assembly site. For example, figure 2.18 showing relation A and B have a semijoin operation over attribute S# and sent the result to query site, as assembly site, directly with cost 11 (3 data of the projection sending to site B + 6 data sending to query site + 2 data of the semijoin result sending to query site = 11 data).

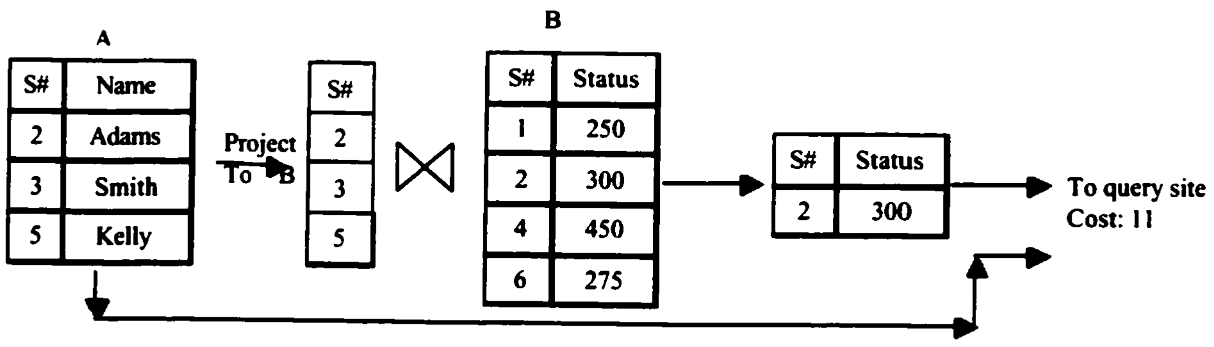


Figure 2.18 Semijoin Result Send to Query Site Directly

If we sent the semijoin result back to site A, at assembly site, and sent the final result of the query to query site, then the cost is reduced to 8 (3 data of the projection sending to site B + 2 data of the semijoin result sending to query site + 3 data of final result sending from site A to query site = 8 data). In figure 2.19, site A is selected as an assembly site. Therefore the semijoin result will sent back to site A and do the join with the relation. Finally, the final result of the query will send to the query site.

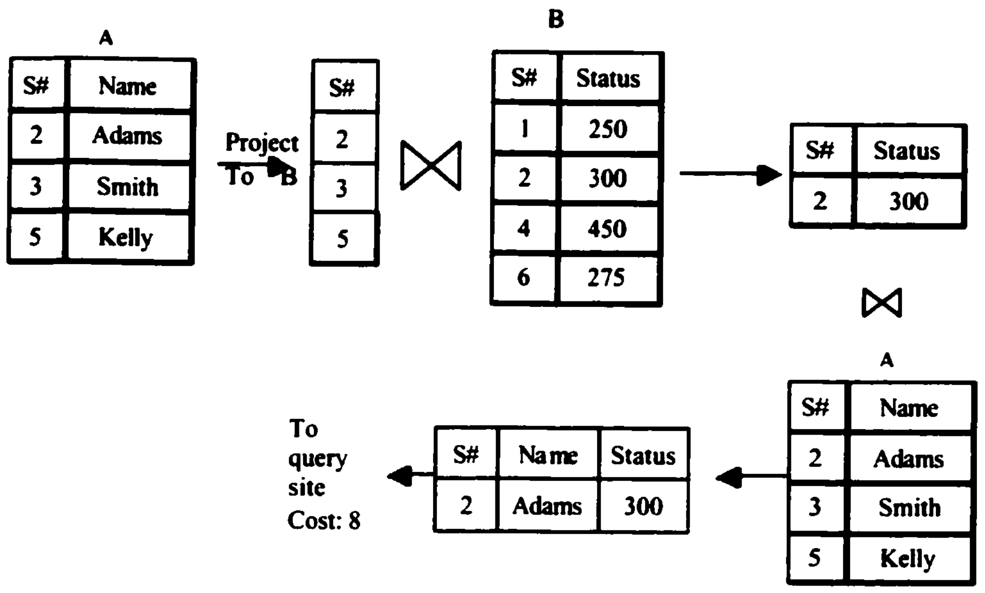


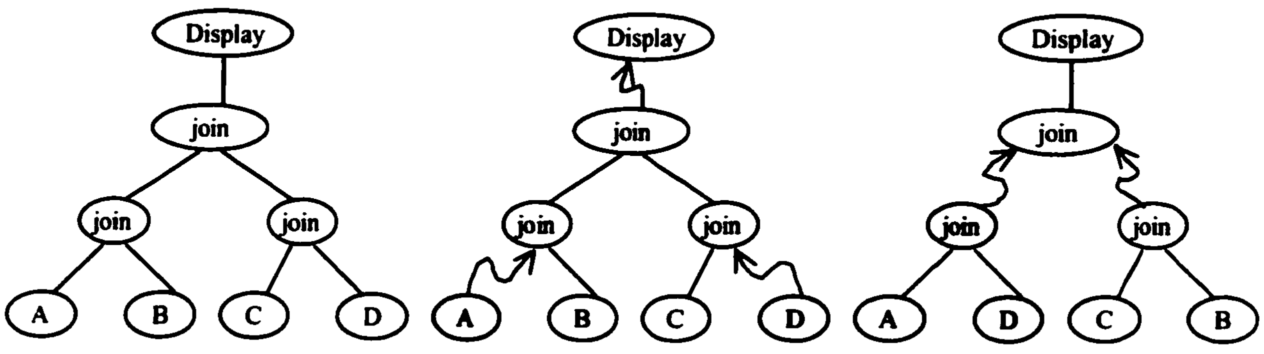
Figure 2.19 Semijoin with Backward Reduction

2.4.3. Two-steps Optimization

Two-Step query optimization reduces the overall complexity of distributed query optimization. It is also useful to exploit caching in a hybrid shipping system because query operators can dynamically be placed at client. For distributed systems, two-step optimization has two basic variants. First, generate a plan that specifies the join order, methods, and access paths during the compile time. Second, it transforms the plan and carries out the site selection to determine where the operator to be executed. Two-step query optimization can result in plans but introduce high communication cost [KOS00].

The join ordering carried out in the first step of two-step optimization is shown in figure 2.20(a). (b) shows site selection in step two of optimization. An optimal plan for the query is show in (c). Relation A and D are located on same site, B and C are located in other site. The query result will be displayed on client. The second plan that employed

2-step optimization, has a higher communication cost because the first step of 2-step optimization was carried out ignoring the location of data and join ordering on communication cost in a distributed system.



a) 2-step plan at compile time b) 2-step plan at run time c) Optimal plan

Figure 2.20 Increased Communication Cost Due to Two-step Optimization

2.4.4. Dynamic Query Re-Optimization

A new triggering approach is proposed in [NWMN99] for improving evaluation performance of long-running queries in distributed databases. In dynamic distributed environment, system configuration and resources availability may be changed during the quires execution period. It causes the query becoming inaccurate. The update statistics information enables to have an up-to-date dynamic cost estimation and query re-optimization for the long-running queries. The basic idea of triggered dynamic re-optimization method is the optimizer can promptly react to change to the dynamic environment during the execution period. The statistical information mainly base on performance instrumentation and performance analysis. Performance instrumentation includes system instrumentation, such as CPU workload and network traffic, and query instrumentation which is the estimation of query characteristics [NWMN99].

In [NWMN99], some triggers are suggested to implement in the system: system triggers, query execution plan (QEP) triggers, operator triggers, and events triggers. They are classified by different system parameters and query characteristics. System triggers are independent of any query plan execution and react with the change of system resources. QEP triggers deal with the changes of query processing environment. Operator triggers include built-in operator triggers and user-defined operator triggers. Built-in operators deal with the changes of usage of library operators in the system. User-defined operator triggers allow the dynamic optimizer to re-optimize the QEP. Event triggers handle the events that defined in performance detector and/or the events issued by external administrator.

The components of the query processing with dynamic triggered re-optimization are described in [NWMN]. The detector dynamically monitor system parameters and query characteristics and sent information to trigger manager. The query agents monitor the external re-optimization command and report it to trigger manager. The trigger manager is responsible for the synchronization of the execution of different trigger events and performance measurement. The dynamic optimizer considers QEP configuration and re-configuration that instructed by trigger manager.

The triggering approach used to re-optimize the query execution plan in the dynamic distributed environment. The triggers monitor system and query changes and report to triggers to organize the re-optimization during the execution of a long-running query plan. In consequence, it increases the accuracy of queries.

Chapter 3. Improvement Algorithm for Semijoin Query Program

3.1. Introduction

In distributed database systems, there are three steps/processes that are commonly recognized; initial local processing, semijoin pre-processing, and final join processing [AHY83, BER81, BL82]. Local processing includes the operations of selection and/or projection. One of the important operations in this area is semijoin which involves joins between relations in different databases that are located in distributed sites. Semijoin also helps to reduce the size of the final relation. The result(s) from the local process will be sent to the query site to perform the final join to produce the query result. In distributed database system, the number of queries will be increased exponentially by the number of relations, attributes, and distributed sites. Therefore, finding an optimal query in this environment is well recognized as an NP-hard problem [CL84, OV99] and heuristic algorithms are commonly used to handle an optimal query problem. There are some heuristic algorithms proposed to provide an efficient and effective, optimal or close to optimal, solution for the optimal query problem in distributed database systems. [BGW+81] and [AHY83] are well known heuristic algorithms in this area. However, those algorithms cannot guarantee the optimality of the solution due to the complexity of the problem.

3.2. Description of Algorithms

In [CL84], four optimality properties are identified and used to check the optimality of a semijoin program for processing tree queries. It also proposed four algorithms to improve a non-optimal semijoin programs to satisfy those four properties and requires the least total data transmission cost to process the query. In [CL84], an optimal semijoin program is represented by an execution graph. An execution graph is a directed acyclic graph whose nodes represent relations (or sites which contain that relation), and whose directed edges represent semijoins. In figure 3.1, the first serial semijoin program and execution graph show that no redundant semijoins occur in the program. The second semijoin program and execution graph show that a non-serial program and include redundant semijoin occurrences.

Semijoin program: R2 → R3, R3 → R2, R2 → R1, R1 → R2, R2 → R3, R3 → R4, R4 → R3, R3 → R2, R2 → R1

Execute graph: R2 → R3 → R2 → R1 → R2 → R3 → R4 → R3 → R2 → R1

Semijoin program: R5 → R2, R2 → R1, R1 → R3, R3 → R1, R1 → R3, R3 → R1, R4 → R1, R1 → R4, R2 → R5, R5 → R1, R2 → R1, R2 → R6

Execute graph:

$$\begin{tikzcd}
& & & & R4 \arrow[dr] & & R4 & \\
R5 \arrow[r] & R2 \arrow[r] \arrow[dr] & R1 \arrow[r] & R3 \arrow[rr] & & R1 \arrow[ur] \arrow[r] & R3 \arrow[r] & R1 \\
& & R5 \arrow[r] & R2 \arrow[urr] \arrow[r] & R6 & & &
\end{tikzcd}$$

Figure 3.1. Execution Graph of Semijoin Program [CL84]

A profitable semijoin occurrence cannot be deleted from the executed graph; otherwise, the transmission cost will be increased. Multiple semijoin occurrences may exist in a semijoin program and some of them are redundant. These redundant occurrences are increasing the total data transmission cost. Those four semijoin optimality properties adopted two rearrangement techniques, early binding and late forking, proposed by Luk and Luk [LL83]. These two techniques used to rearrange the execution sequence of a group of semijoin occurrences in the execution graph and remove the redundant semijoin. The rearrangement helps to reduce the transmission cost. Property 1 and the algorithm are used to identify and remove the redundant semijoin; and then pass it to algorithm 2 for checking the rearrangement property.

The following four properties are used to identify an optimal semijoin program [CL84]:

1. In an optimal program, every semijoin occurrence has to satisfy the reduction set constrain. In other words, there are no redundant semijoins in the optimal program.
2. Each necessary semijoin associated with the execution graph has to be properly embedded in the optimal semijoin program.
3. Each end node of the execution graph of an optimal program must be a final relation.
4. The execution graph of an optimal semijoin program cannot be rearranged by the rearrangement techniques.

In order to understand these four properties, the semijoin program in fgure 3.1, will be used to illustrate each property of an optimal semijoin program. The algorithms proposed in [CL84] will be applied on this semijoin program to obtain an optimal semijoin program with four optimality properties.

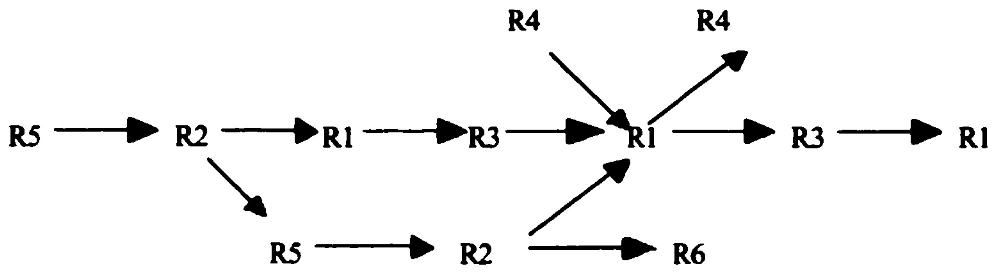


Figure 3.2. Redundant semijoin in a Semijoin Program

Figure 3.2 shows the semijoin program containing some redundant semijoins, R5 → R2, R2 → R1, R1 → R3, and R3 → R1. The repeated semijoins definitely increase the cost/time of the query. According to the first property, algorithm P1 is used to remove the redundant semijoin and reconnect the isolated graph that caused by the deletion.

```
Algorithm P1 [CL84] (* Based on Optimality Property 1 *)
Begin
Check the redundancy of each semijoin occurrence in $p$;
If there exist redundant semijoin occurrences then
  Begin
      Delete all redundant semijoin occurrences and resultant isolated nodes;
      If some node u(x) becomes a start node and not in sequence with some other occurrences of u
        then
        Begin
            delete node u(x) and each edge $u(x) \rightarrow v_i$;
            add $u(x-) \rightarrow v_i$ for each $v_i$ where u(x-) is the previous occurrence of u(x)
        End;
      If some node v(y) becomes an end node and not in sequence with some other occurrences of v
        then
        Begin
            delete node v(y) and each edge $u_i \rightarrow v(y)$;
            add $u_i \rightarrow v(y+)$ for each $u_i$ where v(y+) is the next occurrence of v(y)
        End
  End
End;
```

Removing repeated semijoins might cause the isolation of some other semijoins, such as the example shown in figure 3.3a. Algorithms P1 used to connect the start/end node to its pervious/next occurrence respectively. The resultant graph is shown in figure 3.3b.

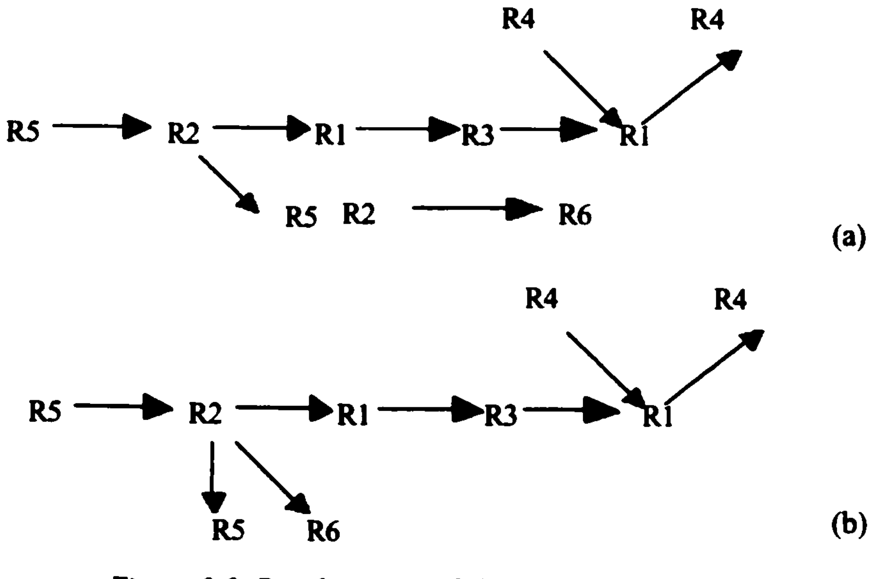


Figure 3.3. Resultant Graph by Algorithm P1

The second property indicates that including all necessary semijoins (NSJ) in leaf-to-root order can result in a full reduction at the root node of the join tree. A necessary semijoin is defined as a backward semijoin from a non-final relation node [CL84]. The following figure shows the corresponding join tree, with two target relations R1 and R2, for the semijoin program in the figure 3.3. According to this property, the join tree including R7 → R5, R5 → R2, R6 → R2, R2 → R1, R3 → R1, and R4 → R1 are NSJ and should be embedded in the optimal semijoin program.

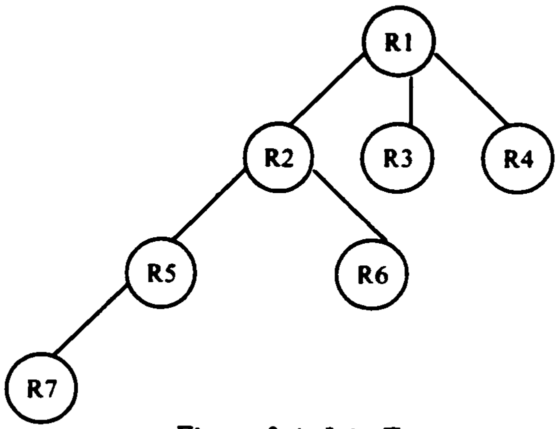


Figure 3.4. Join Tree

Algorithm P3 checks the existence of all NSJ in the semijoin program. If an expected NSJ is absent in the program, then this NSJ will be created and connected to the corresponding relation node. According to the join tree, R7 → R5, and R6 → R2 are missed in this program. Therefore, these two semijoins have been created and integrated into the semijoin program. Figure 3.5 shows the program after the integration.

```
Algorithm P3 [CL84] (* Based on Optimality Property 2 *)
Begin   For k=1 to tree height of Y do
  Begin  If u → v is an NSJ where v is of height k, and it is not properly embedded in p then
        Begin   If no occurrences of u and v are in E(p) then
                        Create u and v, add u → v to E(p);
                If some occurrence of u but no occurrences of v are in E(p) then
                        Create v and add u(l) → v to E(p) where u(l) is the latest occurrence of u;
                If some occurrences of v but no occurrences of u are in E(p) then
                        Create u and add u → v(e) to E(p) where v(e) is the earliest occurrence of v;
                If some occurrence of both u and v are in E(p) then
                        Begin If there exist some occurrences of v not in sequence with u(l) then
                                Choose the earliest such v, say v(i) and add u(l) → v(i) to E(p)
                Else create a new occurrence of v, say v(j) and add u(l) → v(j) to E(p)
        End
  End
End;
```

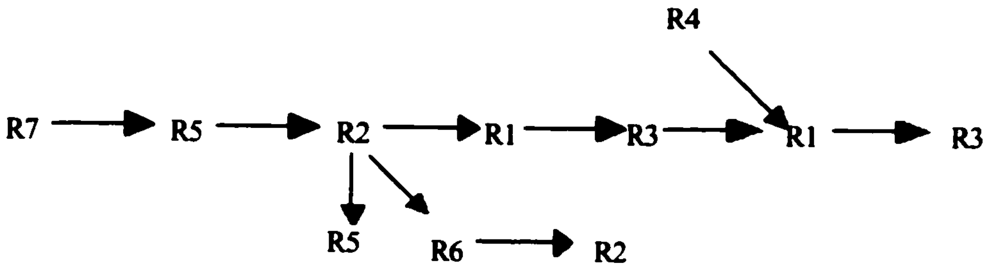


Figure 3.5. NSJ Embedded Semijoin Program

As mentioned before, the target relation in this semijoin program are R1 and R2. Therefore, R1 and R2 should be the end nodes in the execution graph. However, some end nodes in the graph are not the target node. It means that the transmission cost to those nodes are not necessary and should be avoided. This is the main idea of the third property and algorithm P4. It will delete all end nodes and corresponding semijoins which are not final/target relations. Figure 3.6 shows the result of algorithm P4.

```
Algorithm P4 [CL84] (* Based on Optimality Property 3 *)
Begin
If each NSJ associated with Y is properly embedded in p then
   Repeat delete the semijoin occurrence (and the resultant isolated node) whose successor node is an
           end node in E(p) and is a nonfinal relation
   Until every end node in E(p) is a final relation
End;
```

$$\begin{tikzcd}
 & & & & & R4 \arrow[d] \\
R7 \arrow[r] & R5 \arrow[r] & R2 \arrow[r] \arrow[dr] & R1 \arrow[r] & R3 \arrow[r] & R1 \\
 & & & R6 \arrow[r] & R2 &
\end{tikzcd}$$

Figure 3.6. Semijoin Program with Final Relation as End Nodes

The reduction effect of semijoin occurrences is affected by execution sequence in a semijoin program. There are some rearrangement techniques developed in past research works, such as "early binding" and "late forking" proposed by Luk and Luk [LL83], and [BGW+81]. Consider figure 3.7, R5, R6, R2(2) have same successor node R1(2). R1(1) → R5 is upstream of R5 → R1(2), but not R6 → R1(2) and R2(2) → R1(2). R6 → R1(2) and R2(2) → R1(2) have the reduction effect on successor node R1. Therefore, R6 → R1(2) and R2(2) → R1(2) can be propagated earlier by using early binding technique. This technique increases the reduction effect of those semijoins and reduces the total data transmission cost. In the following example, semijoins R6 → R1(2) and R2(2) → R1(2) will be moved from R1(2) to the earlier successor node R1(1).

$$\begin{tikzcd}
R4(1) \arrow[dr] & & & R6 \arrow[dl, shift left] & \\
 & R2(1) \arrow[r] \arrow[dr] & R1(1) \arrow[r] \arrow[ur, shift left] & R5 \arrow[r] & R1(2) \\
R3 \arrow[ur] & & R4(2) \arrow[r] & R2(2) \arrow[ul] &
\end{tikzcd}$$

Figure 3.7. Early Binding

After the rearrangement, there may be cycles in the graph. Therefore, we need to break the cycle in the graph because the cycle is not allowed in an execution graph (see figure 3.8). First, a new node, R1', be created into the graph. Second, remove the edge of R1(1) → R6 and add an edge from R1' to R6. Third, remove all edges that are connected to R1(1) and add new edges to the new node from the corresponding nodes.

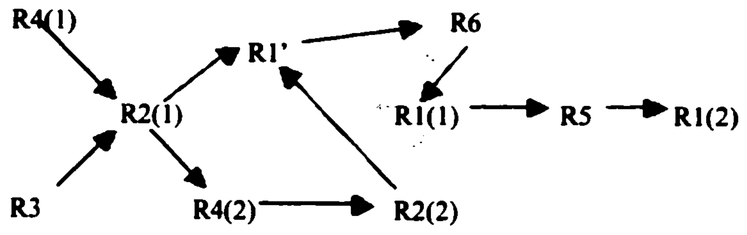


Figure 3.8. Break the cycle in execution graph

The second technique that can be applied in rearrangement is "late forking" which may reduce the total transmission cost for a semijoin program. It will be applied on two edges which have the same predecessor node. Choosing a later occurrence of a relation in the execution graph may have a smaller relation size, if the later occurrence is chosen. Consider the case shown in figure 3.9. R4(2) →R2(2) is downstream from R2(1) → R4(2), but not from R2(1) → R1'. These two semijoins can reduce the relation size of R2. Therefore, removing R2(1) → R1' and adding R2(2) → R1 can reduce transmission cost. After this rearrangement, there is a duplicated edge between R2(2) → R1(1) and one of them can be deleted.

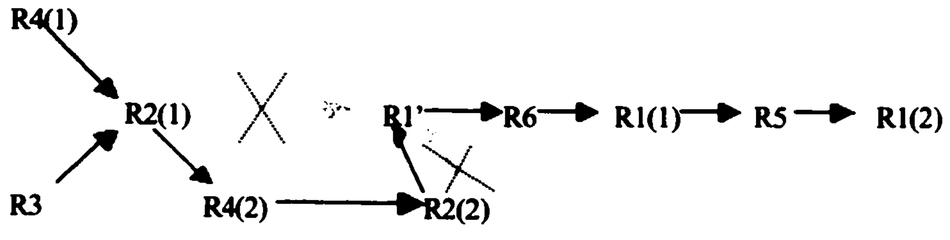


Figure 3.9. Late Forking

```
Algorithm P2 [CL84] (* Based on Optimality Property 4 *)
Begin
While p can be rearranged do
  apply rearrangement techniques
End;
```

According to fourth property, algorithm P2 used to detect the possibility of applying rearrangement techniques on a semijoin program. In the perivous example, it shows that the reduction effect can be increased by applying late forking techniques on R2 → R1 and reduces the transmission cost of R4 → R1 by early binding technique. Therefore, R2 → R1 is moved to the later occurrence of R2 and R4 → R1 is re-located to the early occurrence of R1 (see figure 3.10a). After applying the four algorithms, an optimal semijoin program with four optimal semijoin program properties is produced (see figure 3.10b).

(a)

$$\begin{tikzcd}
 & & & & R4 \arrow[d] & & R4 \arrow[dl, "/" marking] \\
R7 \arrow[r] & R5 \arrow[r] & R2 \arrow[dr] & \times & R1 \arrow[r] & R3 \arrow[r] & R1 \\
 & & & R6 \arrow[r] & R2 \arrow[u] & &
\end{tikzcd}$$

(b)

$$\begin{tikzcd}
 & & & & & R4 \arrow[dr] & & & \\
R7 \arrow[r] & R5 \arrow[r] & R2 \arrow[r] & R6 \arrow[r] & R2 \arrow[rr] & & R1 \arrow[r] & R3 \arrow[r] & R1
\end{tikzcd}$$

Figure 3.10. Rearrangement Techniques and Final Optimal Semijoin Program

The other main concept introduced in [CL84] is "join in two steps". The main idea is to process joins at both step 2 and 3. This approach process the non-final joins in the pre-processing stage, executes a semijoin program which properly embeds all necessary semijoins. The effects of all non-final joins will be achieved at final relations after a semijoin program. The result will be sent to the final site for performing the final join to produce the query result. In the traditional one step approach, all joins are performed at the final site. The forward and backward semijoins are also considered in this approach.

Chapter 4. Improvement Algorithm Analysis

4.1. Rearrangement Techniques and Including all NSJ

As described in the last chapter, rearrangement techniques are used to rearrange the execution sequence of semijoin occurrences in a semijoin program. The goal of the rearrangement is to improve the performance of the program. In other words, it tries to reduce the transmission cost by arranging the execution sequence to obtain an efficient semijoin. In some case studies, an individual semijoin or a segment of a semijoin program may provide the cost reduction after the re-arrangement/algorithms; however, the total response time may be increased.

Considering the example in figure 4.1, statistical information in the table is used to calculate the cost of the semijoin program in this example. Figure (a) repersents the semijoin program before applying improvement algorithms; the response time is 784. Figure (b) shows the first step of improvement algorithms for removing the redundant semijoins. Figure (c) contains the semijoin program before applying the last improvement algorithm for rearranging the semijoin program which has a response time of 975. Figure (d) indicates the rearrangement introducing benefit to semijoins R1 → R3 and R2 → R1. However, the total response time is increased to 1103. In an ideal case, the rearrangement of the executing sequence of semijoins should provide a benefit to the whole semijoin program. However, the rearrangement may only benefit a particular semijoin segment in the program. In the meantime, it also reduces the reduction power of some other semijoins while expecting the new benefit plan to overcome the undesired cost increment.

Relation	Size	Selectivity
R1	150	0.2
R2	100	0.3
R3	300	0.5
R4	50	0.4
R5	400	0.6
R6	200	0.1
R7	500	0.7

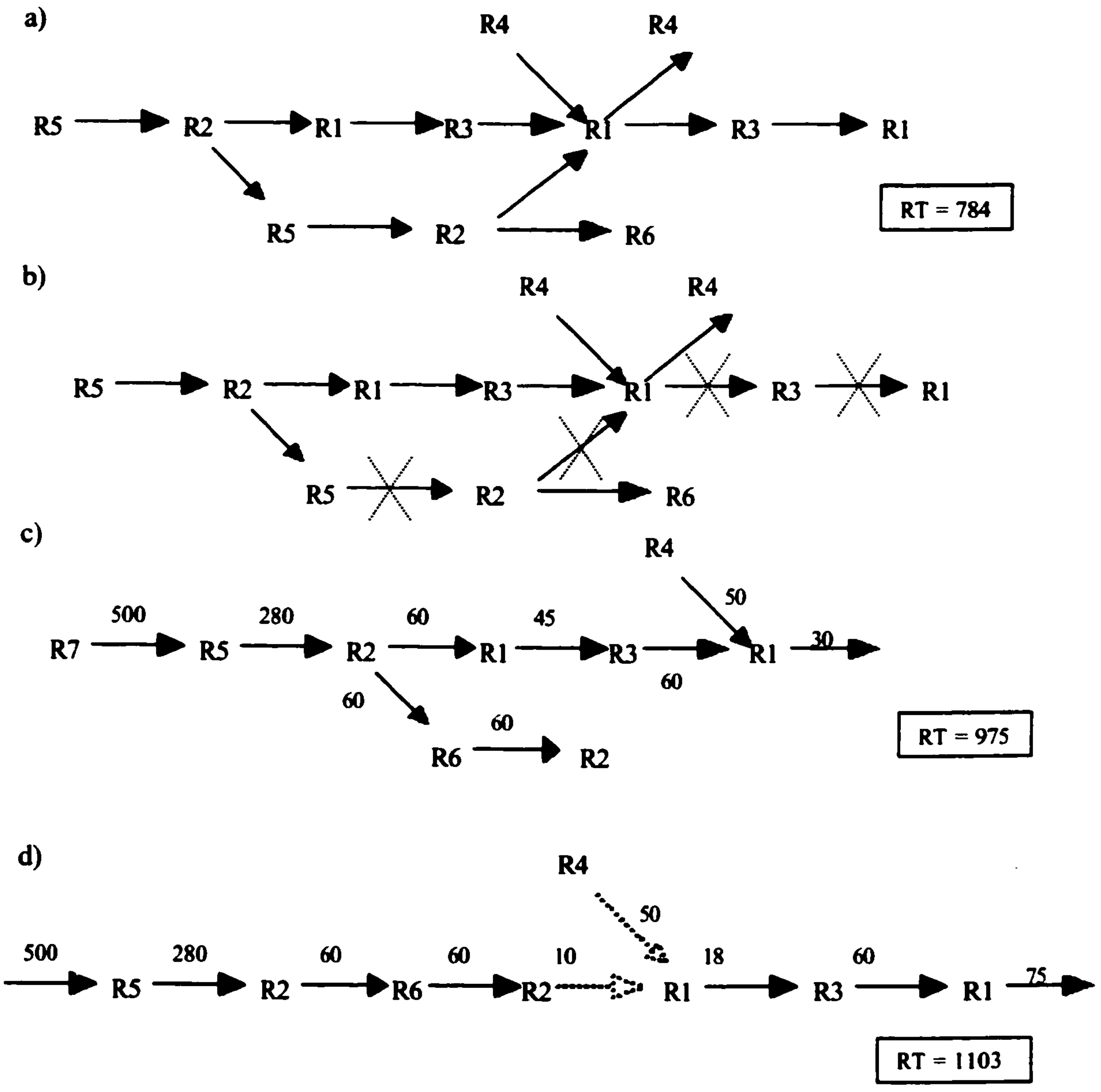


Figure 4.1 Non-beneficial Rearrangements

The improvement algorithms also suggest that a semijoin program should include all NSJ in the join tree. However, it may include some non-profitable semijoins in the program for answering a query. For example, another semijoin query use the same statistical information and join tree as the pervious example, and tries to retrieve records in R2, target relation. Figure 4.2a, is one possible semijoin program for this query but does not satisfy all optimal semijoin program properties. It has 609 units of cost to answer the query. But according to the optimal semijoin program properties, all NSJ should be included in the program. Therefore, semijoin R7 → R5 should be embedded in the semijoin program and it results in an increment of the cost, from 609 to 989 (see figure 4.2b). Therefore, including all NSJs may not be beneficial to the program.

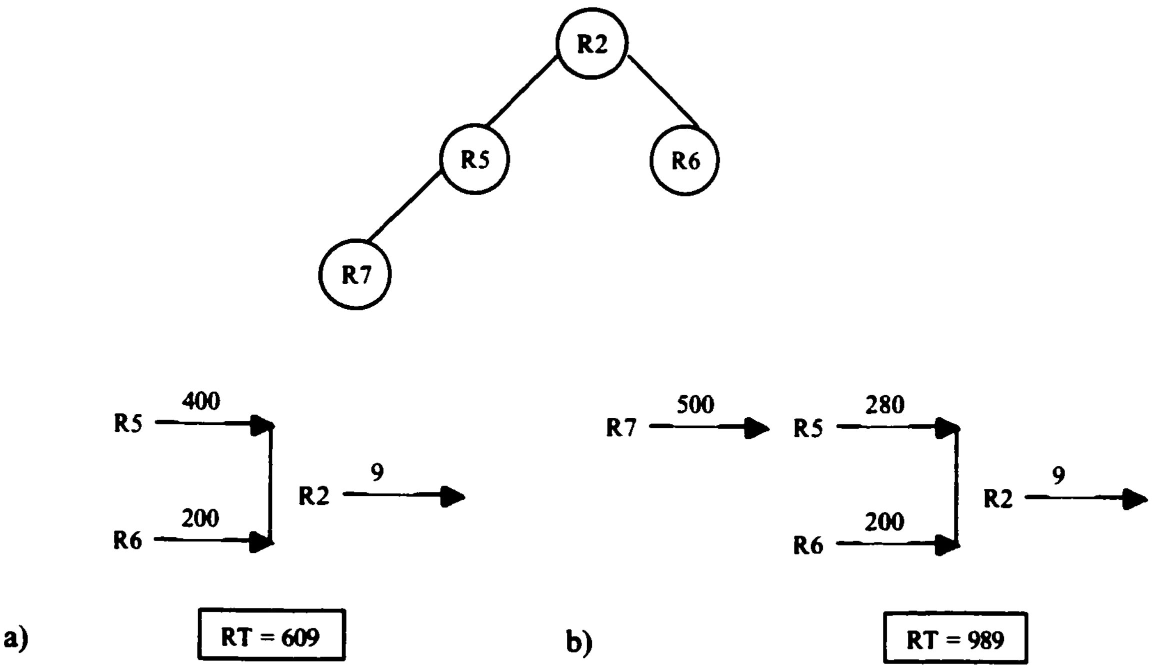


Figure 4.2 Non-beneficial NSJ

4.2. Benefit from Other Operator

The improvement algorithms assumed only one joining attribute in each relation and ignored the reduction effect of other non-joining attributes (as the example in figures 2.12 and 2.13). However, many distributed queries may involve multiple semijoins having common/multiple target relations. Therefore, there may be a cost benefit in performing two semijoins as one composite semijoin, in which the two joining attributes are treated as one composite joining attribute. The reduction effect will be amplified by the composite joining attribute. It also enables a composite semijoin to eliminate some tuples that cannot be accomplished by a single joining attribute. The composite semijoin is proposed in [PC90] to handle these problems by combining multiple semijoins to be a composite semijoin, as described in chapter 2. The main idea of a composite semijoin is to posit a semijoin with a projection of multiple attributes involved in some operations with the same source and destination sites.

Consider the following table and figure which show a composite semijoin providing much more benefit than a regular semijoin. In this example, the response time for semijoin operation is 800 (see figure 4.2a) and composite semijoin is 661 (see figure 4.2b). There is a significant improvement of response time by using a composite semijoin operator for this query. In [PC90], the researcher applied a composite semijoin on many versions of algorithm AHY, W, and S [AHY83, PER85, BGW+81] and discussed many examples and simulation experiments to support this argument. Extending the idea from [PC90], if one applies improvement algorithms to a composite semijoin querying program then there is greater chance of obtaining an optimal – or close to optimal - query program.

It is very easy to adopt a composite semijoin to replace a semijoin as they share the same basic characteristics of semijoin.

		Semijoin						Composite Semijoin						
R		S_i	b_{i1}	p_{i1}	b_{i2}	p_{i2}		S_i	b_{i1}	p_{i1}	b_{i2}	p_{i2}	b_{ic}	p_{ic}
R1		1000	400	0.4	100	0.2		1000	400	0.4	100	0.2	600	0.0012
R2		2000	400	0.4	450	0.9		2000	400	0.4	450	0.9	966	0.00193
R3		3000	900	0.9	---	---		3000	900	0.9	---	---	---	---
R: Relation		S_i: Relation size				b: Attribute size			p: Selectivity					

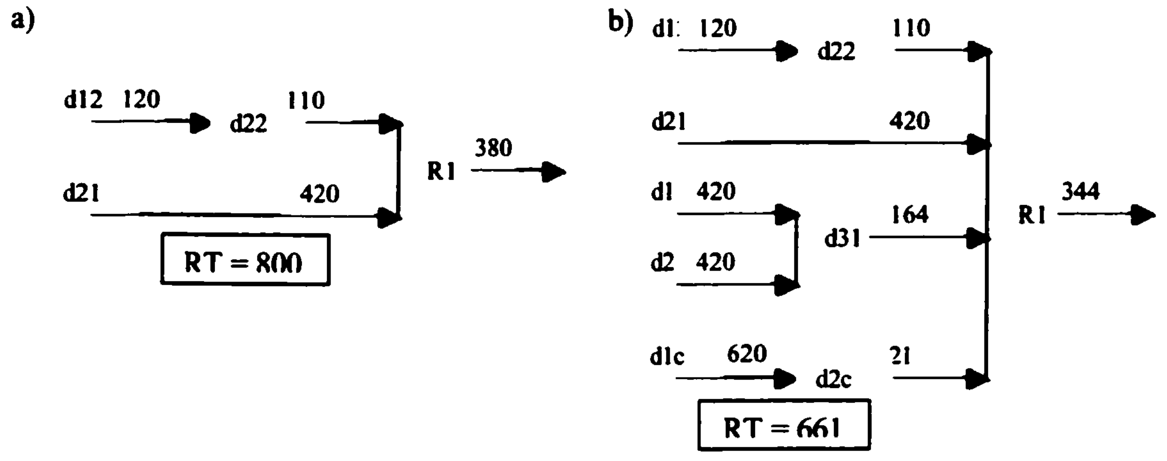


Figure 4.2 Benefit of Composite Semijoin [PC90]

4.3. Serial And Parallel Semijoin Programs

Distributed query processing is defined as the retrieval of data from different sites in a network. The difference between query processing in a centralized database and a distributed database is the potential for decomposing a query into sub-queries which can be processed in parallel. The intermediate results can also be sent in parallel to the required node in a distributed database environment [AHY83]. The decomposition introduced the independence of each paralleled segment of a program. During the processes of the improvement algorithms, it is possible to add and/or remove, and rearrange semijoins in a program to reduce the cost. However, these changes may only provide benefit to a particular segment in a parallel program, but not to the total response time of the whole program. In a serial semijoins program, semijoins are executed one by one. Therefore, adding or removing a semijoin causes a significant effect on the serial program rather than parallel program. Moreover, the independence also reduces the effect of rearrangement techniques in a parallel semijoin program. In some situations, if total time instead of response time is the main concern of a query, then fulfilling the properties of an optimal semijoin program definitely helps to reduce the total time. Removing redundant semijoins and rearranging the execution sequence are two key operations for reducing total time in a semijoin program. The following figure shows the cost reduction power of removing redundant semijoins in a parallel program. In figure 4.2a, it shows an original parallel semijoin program with the total time, 20123.7. After applying improvement algorithms, the total time is reduced to 8719.67 and shown in figure 4.2b. There is a 57% improvement on the total time. The result also indicated many non-optimal semijoin programs including a large amount of redundant semijoins, especially those occurring happening in the program composed by a large number of relations and attributes.

a)

```
Schedule R1  Cost: 366.714
 ^
 |
  <--- d43  Cost: 448
  <--- d23  Cost: 482
  <--- d31  Cost: 627
  <--- d51  Cost: 772
  <--- d33  Cost: 115.054
 |      | <--- d43  Cost: 448
 |      | <--- d23  Cost: 482
 |      | <--- d13  Cost: 264.746
 |          | <--- d43  Cost: 448
 |          | <--- d23  Cost: 482
  <--- d53  Cost: 50.5509
 |      | <--- d43  Cost: 448
 |      | <--- d23  Cost: 482
 |      | <--- d13  Cost: 264.746
 |          | <--- d43  Cost: 448
 |          | <--- d23  Cost: 482
 |      | <--- d33  Cost: 115.054
 |          | <--- d43  Cost: 448
 |          | <--- d23  Cost: 482
 |          | <--- d13  Cost: 264.746
 |                | <--- d43  Cost: 448
 |                | <--- d23  Cost: 482
  <--- d63  Cost: 28.4585
 |      | <--- d43  Cost: 448
 |      | <--- d23  Cost: 482
 |      | <--- d13  Cost: 264.746
 |            | <--- d43  Cost: 448
 |            | <--- d23  Cost: 482
 |      | <--- d33  Cost: 115.054
 |            | <--- d43  Cost: 448
 |            | <--- d23  Cost: 482
 |            | <--- d13  Cost: 264.746
 |                | <--- d43  Cost: 448
 |                | <--- d23  Cost: 482
 |      | <--- d53  Cost: 50.5509
 |            | <--- d43  Cost: 448
 |            | <--- d23  Cost: 482
 |            | <--- d13  Cost: 264.746
 |                | <--- d43  Cost: 448
 |                | <--- d23  Cost: 482
 |            | <--- d33  Cost: 115.054
 |                | <--- d43  Cost: 448
 |                | <--- d23  Cost: 482
 |                | <--- d13  Cost: 264.746
 |                     | <--- d43  Cost: 448
 |                     | <--- d23  Cost: 482
  <--- d54  Cost: 951
  <--- d44  Cost: 1014

Total time = 20123.7
Total Response time = 1380.71
```

b)

```
Schedule R1  Cost: 366.714
 ^
 |
  <--- d43  Cost: 448
  <--- d23  Cost: 482
  <--- d31  Cost: 627
  <--- d51  Cost: 772
  <--- d33  Cost: 115.054
 |      |<--- d43  Cost: 448
 |      |<--- d23  Cost: 482
 |      |<--- d13  Cost: 264.74
  <--- d53  Cost: 50.5509
 |      |<--- d43  Cost: 448
 |      |<--- d23  Cost: 482
 |      |<--- d13  Cost: 264.74
 |      |<--- d33  Cost: 115.05
  <--- d63  Cost: 28.4585
 |      |<--- d43  Cost: 448
 |      |<--- d23  Cost: 482
 |      |<--- d13  Cost: 264.74
 |      |<--- d33  Cost: 115.05
 |      |<--- d53  Cost: 50.550
  <--- d54  Cost: 951
  <--- d44  Cost: 1014

Total time = 8719.67
Total Response time = 1380.71
```

Figure 4.2 Parallel Semijoin Program

Chapter 5. Evaluation

5.1. Performance of Improvement Algorithms

In order to study the performance and characteristics of the improvement algorithms, the algorithms have been implemented, tested and evaluated. The relation generator that developed by the database research group in the University of Windsor is used to generate the statistical data for forming a semijoin program. The generator is modified to fit the requirements of improvement algorithms and this thesis. The statistical data includes the relation size, attribute size, and attribute selectivity. The semijoin program could be generated by any algorithm for tree query. In this thesis, the AHY algorithm (respond time version) [AHY83] has been chosen to generate a set of semijoin programs as input of the improvement algorithms from the statistical data. The AHY algorithm already has been described in chapter 2. It is used for the query optimization in a distributed environment. It means that the semijoin program provided by this algorithm already contains an optimal or near-to optimal semijoin program. Therefore, it provides a very suitable testing sample for improvement algorithms [CL84] to test the optimality of a semijoin program and to improve non-optimal semijoin programs. The evaluation focuses on the change of response time and total time of a semijoin program before and after applying improvement algorithms.

The relation generator has generated a large set of statistical data tables for the testing. Tables are andomly generated and based on the range of selectivity to be categorized in four groups, 0.100 – 0.3669, 0.367 – 0.6429, 0.643 – 0.901, and 0.100 – 0.901. Two extreme bias ranges, 0.000 – 0.009 and 0.901 – 0.999, are not considered. The combination of relations and attributes are limited to (3 – 6) relations and (2 – 4) attributes. The attribute size domain is from 500 to 1500 and relation size domain is from 500 to 6000. The generator produced 200 tables for each combination. Therefore, there are 9600 (4 relations x 3 attributes x 200 tables x 4 selectivity segments) tables that have been generated. It provides a total of 43200 (10800 schedules x 4 selectivity segments) semijoin schedules/programs to test the performance of improvement algorithms. The following figures (figure 5.1 – 5.4) are the statistical summary for the testing result. Each figure represents one particular segment of selectivity.

Selectivity 0.100-0.3669	2 Attributes Total time	2 Attributes Response time	3 Attributes Total time	3 Attributes Response time	4 Attributes Total time	4 Attributes Response time
	(600 Relation Schedules for Each Combination)					
3 Relations						
Increased	72	61	52	46	63	54
%	12	10.16667	8.666667	7.666667	10.5	9
Decreased	0	0	0	0	0	0
%	0	0	0	0	0	0
	(800 Relation Schedules for Each Combination)					
4 Relations						
Increased	144	169	129	138	108	116
%	18	21.125	16.125	17.25	13.5	14.5
Decreased	127	0	69	0	64	0
%	15.875	0	8.625	0	8	0
	(1000 Relation Schedules for Each Combination)					
5 Relations						
Increased	164	233	178	198	113	148
%	16.4	23.3	17.8	19.8	11.3	14.8
Decreased	318	0	151	0	206	0
%	31.8	0	15.1	0	20.6	0
	(1200 Relation Schedules for Each Combination)					
6 Relations						
Increased	149	271	138	203	150	232
%	12.41667	22.58333	11.5	16.91667	12.5	19.33333333
Decreased	417	0	315	0	253	0
%	34.75	0	26.25	0	21.08333	0

Total Number of Relation Schedules: 10800

Total % Increased Total Cost: 13.52% Increased Response time: 17.31%

Decreased Total time: 17.18% Decreased Response time: 0%

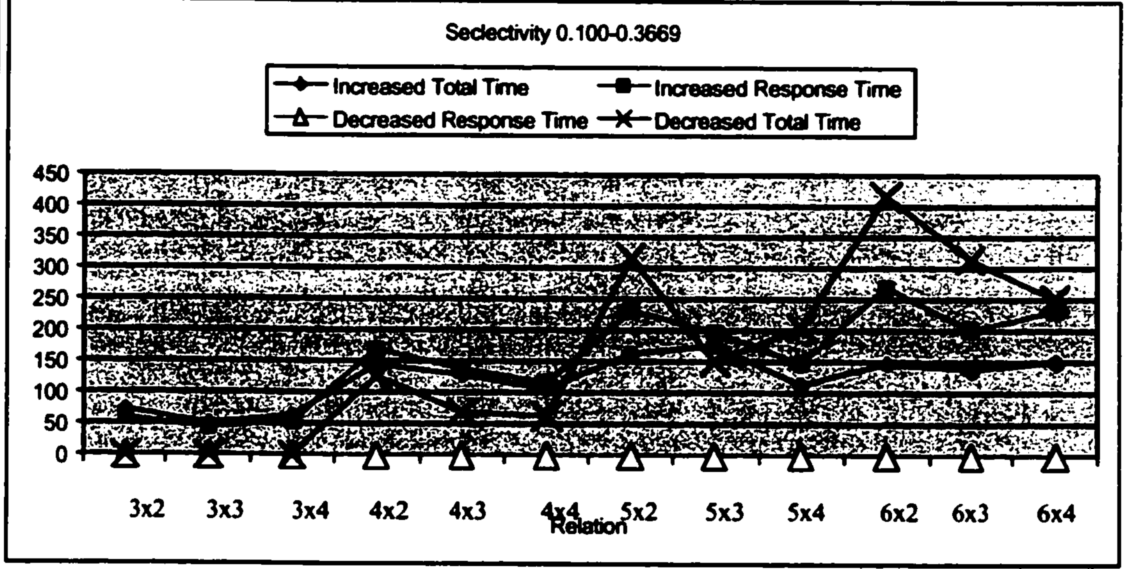


Figure 5.1 Testing Result with Selectivity 0.100-0.3669

Selectivity 0.367-0.6429	2 Attributes Total time	2 Attributes Response time	3 Attributes Total time	3 Attributes Response time	4 Attributes Total time	4 Attributes Response time
	(600 Relation Schedules for Each Combination)					
3 Relations						
Increased	0	2	0	4	0	2
%	0	0.333333	0	0.666667	0	0.333333333
Decreased	0	0	0	0	0	0
%	0	0	0	0	0	0
	(800 Relation Schedules for Each Combination)					
4 Relations						
Increased	0	5	0	2	0	2
%	0	0.625	0	0.25	0	0.25
Decreased	132	0	104	0	95	0
%	16.5	0	13	0	11.875	0
	(1000 Relation Schedules for Each Combination)					
5 Relations						
Increased	0	2	0	5	0	1
%	0	0.2	0	0.5	0	0.1
Decreased	475	0	355	0	299	0
%	47.5	0	35.5	0	29.9	0
	(1200 Relation Schedules for Each Combination)					
6 Relations						
Increased	0	12	0	2	0	3
%	0	1	0	0.166667	0	0.25
Decreased	792	0	670	0	577	0
%	66	0	55.83333	0	48.08333	0

Total Number of Relation Schedules: 10800

Total % Increased Total Cost: 0% Increased Response time: 0.39%

Decreased Total time: 32.40% Decreased Response time: 0%

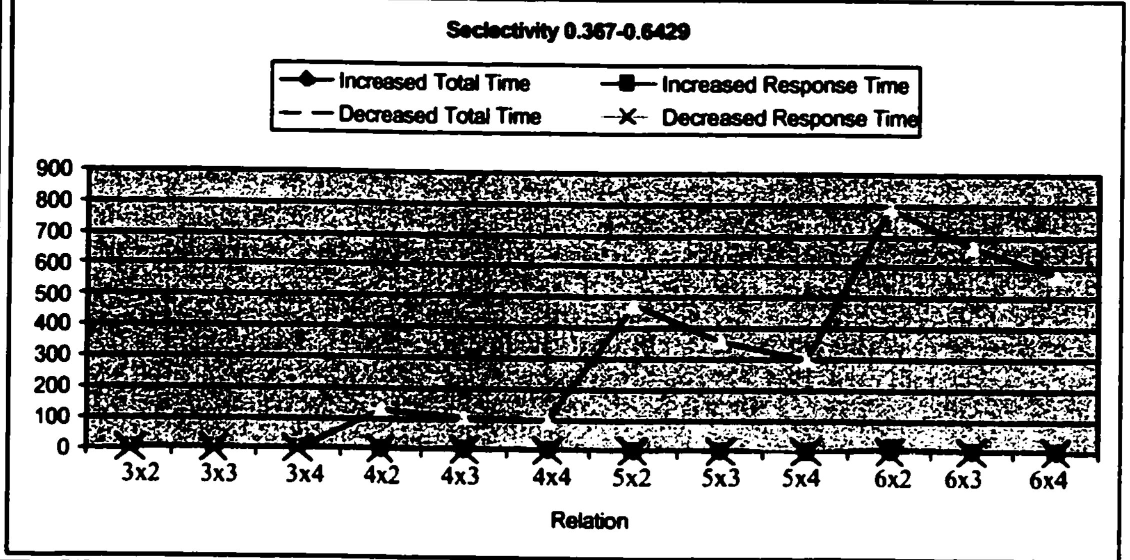


Figure 5.2 Testing Result with Selectivity 0.367-0.4629

Selectivity	2 Attributes		3 Attributes		4 Attributes	
0.643-0.901	Total time	Response time	Total time	Response time	Total time	Response time
	(600 Relation Schedules for Each Combination)					
3 Relations						
Increased	0	0	0	1	0	2
%	0	0	0	0.166667	0	0.333333333
Decreased	0	0	0	0	0	0
%	0	0	0	0	0	0
	(800 Relation Schedules for Each Combination)					
4 Relations						
Increased	0	0	0	0	0	0
%	0	0	0	0	0	0
Decreased	70	0	35	0	63	0
%	8.75	0	4.375	0	7.875	0
	(1000 Relation Schedules for Each Combination)					
5 Relations						
Increased	0	3	0	2	0	4
%	0	0.3	0	0.2	0	0.4
Decreased	178	0	144	0	162	0
%	17.8	0	14.4	0	16.2	0
	(1200 Relation Schedules for Each Combination)					
6 Relations						
Increased	0	2	0	3	0	2
%	0	0.166667	0	0.25	0	0.166666667
Decreased	421	0	363	0	341	0
%	35.08333	0	30.25	0	28.41667	0

Total Number of Relation Schedules: 10800

Total % Increased Total Cost: 0% Increased Response time: 0.18%

Decreased Total time: 16.45% Decreased Response time: 0%

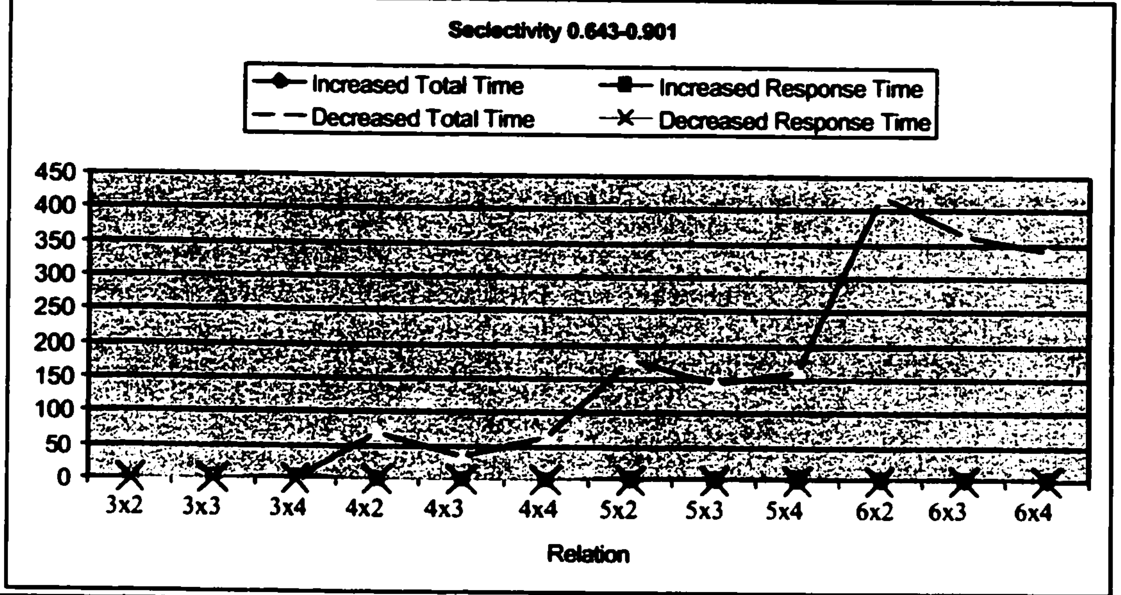


Figure 5.3 Testing Result with Selectivity 0.643-0.901

Selectivity 0.100-0.901	2 Attributes Total time	2 Attributes Response time	3 Attributes Total time	3 Attributes Response time	4 Attributes Total time	4 Attributes Response time
	(600 Relation Schedules for Each Combination)					
3 Relations						
Increased	53	50	59	48	61	51
%	8.833333	8.333333	9.833333	8	10.16667	8.5
Decreased	0	0	0	0	0	0
%	0	0	0	0	0	0
	(800 Relation Schedules for Each Combination)					
4 Relations						
Increased	138	169	113	130	140	125
%	17.25	21.125	14.125	16.25	17.5	15.625
Decreased	126	0	76	0	54	0
%	15.75	0	9.5	0	6.75	0
	(1000 Relation Schedules for Each Combination)					
5 Relations						
Increased	173	235	144	174	158	172
%	17.3	23.5	14.4	17.4	15.8	17.2
Decreased	288	0	175	0	133	0
%	28.8	0	17.5	0	13.3	0
	(1200 Relation Schedules for Each Combination)					
6 Relations						
Increased	146	259	154	231	166	215
%	12.16667	21.58333	12.83333	19.25	13.83333	17.91666667
Decreased	419	0	337	0	269	0
%	34.91667	0	28.08333	0	22.41667	0

Total Number of Relation Schedules: 10800

Total % Increased Total Cost: 13.94% Increased Response time: 17.21%

Decreased Total time: 17.38% Decreased Response time: 0%

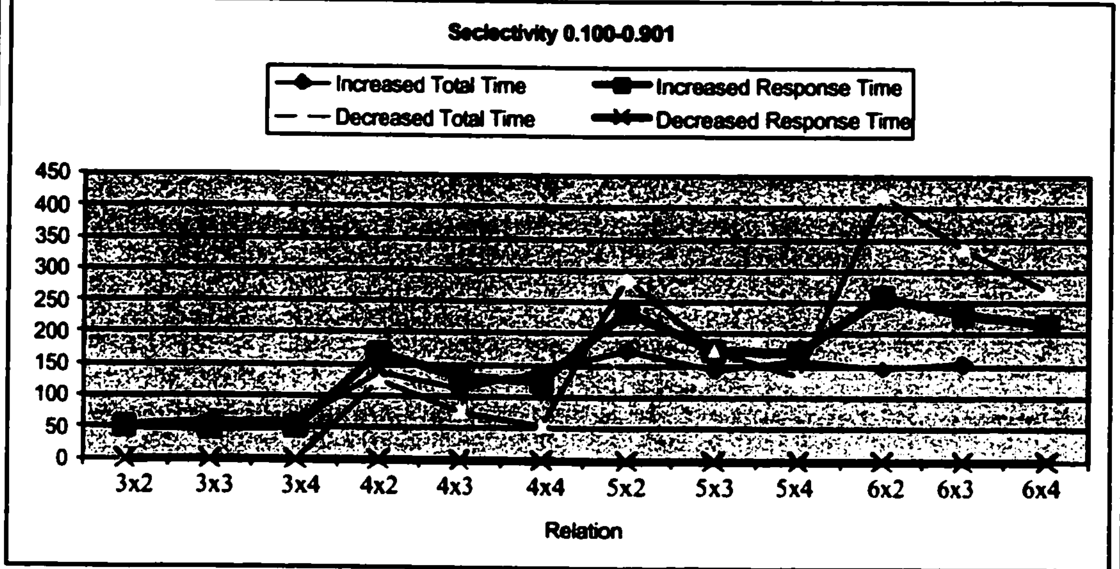


Figure 5.4 Testing Result with Selectivity 0.100-0.3669

The testing results are summarized in four groups according to their selectivity range. The results indicated that there is no improvement on the response time in any parallel semijoin program. On the other hand, improvement algorithms provided a significant improvement on total time in most schedules but not the schedules with a small number of relations and attributes, such as 3 x 2 and 3 x 3.

In the first segment of the selectivity (0.100 – 0.3669), there are 17.78% of total 10800 schedules improved on total time. However, 13.52% of schedules have increased cost of total time and 17.31% of schedules have extended their response time. On average, the improvement of total time follows the increment of the number of relations and attributes (see figure 5.1).

The improvement algorithms have the best performance in the second selectivity segment (0.364 – 0.6429). The algorithms reduced the total time of 32.4% of schedules. The algorithms almost did not increase the cost of total time and response time. In fact, only 0.39% of schedules increased response time (see figure 5.2). The behaviour in the third segment (0.643 – 0.901) is very similar to the previous segment. It has half the number of all results of the second segment. 16.45% of schedules have total time reduced and 0.18% of schedules are increased the response time (see figure 5.3).

The last group of schedules are distributed on the whole range of selectivity from 0.100 to 0.901. The performance is concluded in the same way of the first segment. The increased total time of the schedules has been shown in 13.94% of the population.

17.38% of schedules decreased total time. 17.21% of schedules increased in response time (see figure 5.4).

According to the experiment results, the performance of the schedules which within the selectivity range of 0.100 – 0.3669 is frequently oscillated but there is no effect of improvement algorithms in other selectivity segments, except the improvement on total time. The significant improvement on the total time of a parallel semijoin program is caused by the semijoin formation structure that provided by AHY algorithm. The final schedule provided by AHY algorithm that accumulated the some candidate schedules in parallel. Therefore, it contains many redundant semijoins that can be eliminated by improvement algorithms. On the other hand, the independency between the sub-schedules/sub-programs minimized the improvement performance of rearrangement techniques. Therefore, the execution orders of sub-schedules do not cause a big impact on response time of the whole semijoin program. The natural characteristics of AHY semijoin program are the main reason of the improvement algorithms' power cannot be fully demonstrated. Although, the response time is not improved at all in the experiments but the optimality properties of AHY semijoin program have been verified.

5.2. Implementation Approaches of Improvement Algorithms

Message Passing Interface (MPI)

Message passing is a paradigm used widely on certain classes of parallel machines and distributed memory system, including workstation clusters and heterogeneous networks. MPI includes a large set of collective communication operations, virtual topologies, and different communication modes. In a rapidly changing environment of high performance computer and communication technology, portability is one of the most important aspects of computer application development. Programs using MPI runs on any platform, which has a MPI implementation without any need to modify the codes. The programs are independent of machine architecture and type of network employed to transfer data from one processor to another. Distributed and/or parallel computing models also increase the computational power and performance for the system. In ideal case, the system with n computers/processors is expected to have n times computational power/performance.

In distributed database systems, databases are distributed on a computer network. This configuration allows distributed query processing to take the advantages that provided by MPI model. Each node/relation site in the system can take the responsibility for one relation or its own relation of the queries (see figure 5.5).

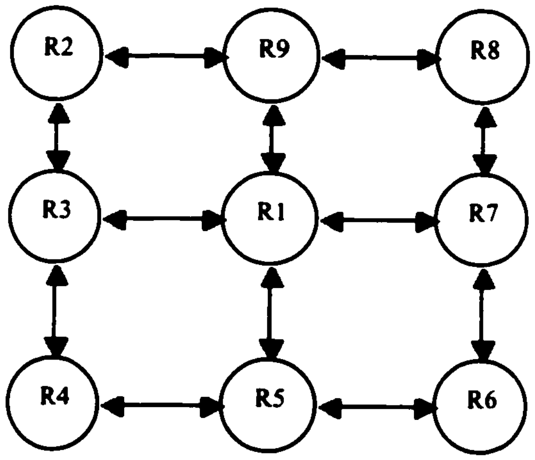


Figure 5.5 MPI Approach for Distributed Query Processing

This approach maximizes the computational power usage of the system and provides better performance for executing a query. In the experiment for this approach, a parallel/multi-processors system is used to simulate a distributed environment, and handle the production of semijoin programs and optimization by applying improvement algorithms. There are 100 batches of semijoin program, each batch contains 43,200 schedules/programs, have been tested. The unsurprised testing result shows that the average execution time of a query optimization in the system is reduced from 105 seconds running on one processor to 40 seconds with 4 processors. The result does not match the ideal case because the communications between processors increase the cost. The implementation is very easy to be modified and fit into real distributed environment because the MPI model supports both systems. And also, it can be extended to support dynamic query optimization. The figure also shows that the nodes/processors can communicate with its neighbours. Each processor can transfer the most updated information of each relation during the optimization. Therefore, it is possible to provide much more precise query plan dynamically.

Nested Improvement Approach

One-shot approach of improvement algorithms is to apply the improvement algorithms on a semijoin program without the consideration of its size and complexity. However, handling a simple and smaller semijoin program is easier and faster than a bigger and complex one by intuition. Therefore, the nested approach is proposed in this thesis and expected to provide benefit for improvement algorithms in terms of execution time. This approach applies the improvement algorithms on semijoin sub-programs during the semijoin program production of AHY algorithm. Improvement algorithms optimize the sub-programs, and try to provide optimized sub-programs for the final program integration. It reduces the chance of optimization for the final semijoin program. The experiment result shows the implementation of this approach has same average execution time, 105 seconds, as one-shot approach. The cause of the disappointment is that the nested approach implementation re-uses the original one-shot approach program to check the optimality of the final semijoin program. The original one-shot approach program checks the optimality of all sub-programs. Nested approach should skip this step because the sub-programs are already optimized. Therefore the implementation of nested approach should not re-use the one-shot approach program for checking the optimality of the final semijoin program. It is believed that the nested approach still able to provide benefit for the improvement algorithms on a semijoin program.

Chapter 6. Conclusion and Future Work

Query optimization in distributed database system becomes an NP-hard problem due to the number of alternative queries is increased exponentially by the number of database sites, relations, and attributes. This fact is well recognized in this research area. Therefore, some researchers have proposed many heuristic algorithms in the past, such as AHY and SDD-1. All these heuristic algorithms are trying to provide an optimal or near-to optimal solution for answering a query. However, there is no algorithm which can guarantee the optimality of a query due to the complexity of the problem and the lack of properties of an optimal query. This reason motivated some researchers to do some researches for identifying the optimality properties of semijoin query program. The properties and corresponding algorithms had been proposed in [CL84] but there is no publication which has been found for evaluating the performance and limitation of these properties and algorithms in accessible resources during the proceeding of this thesis. Therefore, this thesis has done some research studies on the performance and limitation of the improvement algorithms and the properties that proposed in [CL84] for an optimal semijoin query program in a distributed database system.

This thesis takes a semijoin schedules/programs that produced by AHY algorithms (response time version) [AHY83] as the inputs of improvement algorithms for testing its performance and limitation. According to the experiment result, the improvement algorithms cannot provide improvement of performance, in terms of response time for a parallel semijoin program. But algorithms have a significant improvement on total time of parallel program. Although, there is no improvement on response time of the parallel semijoin program in the experiment but the study shows AHY semijoin program already satisfy most of the properties of an optimal semijoin program that was proposed in [CL84]. Based on the characteristics of parallel and serial semijoin program, the study also indicated the execution sequence of semijoin operations is affecting the results of improvement algorithms. The results also depend on the input semijoin program and its size, and the parallelism of the program. The execution independence of paralleled semijoin segments/sub-program reduces the improvement power of the algorithms. On the other hand, the dependence in serial semijoin program enhances the performance of the improvement algorithms.

The other uncertainty bonded with the improvement algorithms is the benefit that provided by rearrangement techniques. The counter example shown in chapter 4 indicated that the rearrangement might only benefit to some particular semijoins. In the mean time, the rearrangements also increase the cost of some other semijoins and hope the new benefit can cover the cost increment on other semijoin.

Every research will set some assumptions to narrow down the scope of the research. However, it may also limit the benefit that provided by the ignored factor. In the improvement algorithms, only one joining attribute is assumed existing in a relation and only semijoin operator has been used. Multiple joining attributes are commonly occurred in the relations of real distributed database systems. These assumptions ignore the benefit that can be provided by other query operators, such as composite semijoin, for multiple joining attributes.

There are couple issues related to improvement algorithms have been discussed in this thesis. Some studies, such as the reduction power of composite semijoin and the dependence in serial semijoin program enhancing the performance of improvement algorithms, are based on the natural characteristics to state their effect in distributed query processing. Some simulations and experiments can be done in the future to consolidate some ideas stated in this thesis.

Besides the performance evaluation, some performance improvement researches also can be constructed in the future. As mentioned in the discussion of composite semijoin in chapter 2, the proposed composite semijoin cannot guarantee that the benefit will be provided. Even, there are lot of examples and simulation showing the reduction power of composite semijoin. However, the research still not guarantees the benefit of composite semijoin. Therefore, it is a good research direction to adopt a combined operation of composite semijoin and traditional semijoin for maximizing and providing confidence of the benefit in the improvement algorithms for semijoin program.

One of the main limitations of the improvement algorithms is the query structure that limited to tree queries only. Therefore, the other very good research direction in distributed query processing is to extend improvement algorithms to handle general queries. One of the challenges behind the general queries is how to handle the cycle in the query/execution graph (or called as cyclic query/graph) and finding an efficient semijoin program [BC81, BG81, GS82].

In past ten years, dynamic query optimization follows the rapidly development of computer network to become much more important. The MPI approach studies show the potential computational power of MPI model. It can be extended for dynamic query optimization to provide an efficient dynamic query plan.

In a semijoin program, if a particular node/relation is a successor node of multiple semijoins then the waiting time for completing all semijoins should be considered, not only focusing on the amount of data to be transferred and/or the time required for the transmission, in finding an optimal semijoin program. In some extreme cases, the waiting cost may be highly over the benefit that provided by a semijoin itself. This situation may also relate to the physical computing network problem. These two issues should be stated in the optimizing process for a non-optimal semijoin program.

Glossary

Backward semijoin: Semijoin directed toward to root node.

Cost: Resource(s), including time and storage, required for answering a query.

Distributed database system: A collection of multiple, logically interrelated databases distributed over a computer network.

Distributed query processing: A process to transform a high-level query to a low-level query for retrieving the data in an efficient and effective way from the distributed database.

Execution time: The time between receiving a command of executing a corresponding program(s)/query(s) and the arrival of the output to the expected destination.

Execution graph: A directed acyclic graph whose nodes represent relations and directed edges represent semijoins.

Final relations: The union of target relations and related relations.

Final join: Both joining relations are final relations.

Forward semijoin: Semijoin directed away from the root node.

Heuristic algorithm: An algorithm will generate solutions which are guaranteed to be close to the optimal solution in polynomial time.

Necessary semijoin (NSJ): A backward semijoin with its predecessor node being a non-final relation.

NP-hard problem: Problems are at least as hard to solve as any NP problem.

NP problem: A set of all decision problems, which can be solved by non-deterministic algorithms in polynomial time.

Optimal query: The best or most efficient query which requires the least total data transmission cost to process the query.

Optimization problem: A problem that seeks for the best solution among many possible solutions, according to a simple cost criterion. It corresponds a decision problem.

Related relations: Relations which are intermediate nodes in the paths between any two target relations.

Response time: Time required for providing information to answer a query

Selectivity: the number of different values occurring in the attribute divided by the number of all possible values of the attributes

Semijoin program: A query only contains a set of semijoin operations and constructed by including all backward semijoins in a breadth first leaf-to-root order.

Total time: The sum of the time of all transmissions required in the schedule.

Tree query: A query embedded in a tree structure.

Tree structure: A data structure whose node can have two or more branches/children. There is no any connection between each branch.

Appendices (Sample Test Data and Result)

- The indicated cost beside an attribute or a relation number means the cost required to transfer that particular attribute or relation.
- Some results of improvement algorithms are same as the schedules that produced by AHY algorithm. It means that no improvement has been occurred on that particular schedules.

Statistical table for 6 relations with 4 attributes:

Rel. size	Att. 1 Size	Select.	Att. 2 size	Select.	Att. 3 size	Select.	Att. 4 size	Select.
3400	132	0.207	149	0.194	197	0.185	0	0.000
1600	0	0.000	0	0.000	0	0.000	336	0.261
1700	108	0.169	230	0.299	0	0.000	227	0.176
1100	0	0.000	188	0.245	349	0.327	0	0.000
3000	128	0.201	0	0.000	0	0.000	0	0.000
2900	0	0.000	206	0.268	0	0.000	187	0.145

AHY Algorithm (Response Time Version) Result:

```
Schedule R1  Cost: 48.2962
 ^
 |
  <--- d31  Cost: 128
  <--- d51  Cost: 148
  <--- d42  Cost: 208

Total time = 532.296
Total Response time = 256.296

************************
Schedule R2  Cost: 60.832
 ^
 |
  <--- d64  Cost: 207
  <--- d34  Cost: 247

Total time = 514.832
Total Response time = 307.832

************************
Schedule R3  Cost: 33.722
 ^
 |
  <--- d51  Cost: 148
  <--- d12  Cost: 169
  <--- d11  Cost: 24.4839
 |      |  <--- d31  Cost: 128
 |      |  <--- d51  Cost: 148

Total time = 651.206
Total Response time = 206.206
```

```
Schedule R4  Cost: 30.5804
 ^
 |
  <--- d12  Cost: 169
  <--- d13  Cost: 217
  <--- d62  Cost: 29.7912
 |      |  <--- d12  Cost: 169
 |      |  <--- d42  Cost: 208
Total time = 823.372
Total Response time = 268.372
************************
Schedule R5  Cost: 124.949
 ^
 |
  <--- d31  Cost: 128
  <--- d11  Cost: 24.4839
 |      |  <--- d31  Cost: 128
 |      |  <--- d51  Cost: 148
Total time = 553.433
Total Response time = 297.433
************************
Schedule R6  Cost: 27.2535
 ^
 |
  <--- d12  Cost: 169
  <--- d42  Cost: 208
  <--- d34  Cost: 247
  <--- d32  Cost: 20.1393
 |      |  <--- d12  Cost: 169
 |      |  <--- d42  Cost: 208
 |      |  <--- d62  Cost: 29.7912
 |             |  <--- d12  Cost: 169
 |             |  <--- d42  Cost: 208

Total time = 1455.18
Total Response time = 285.184
```

Improvement Algorithms Result

```
************************
Schedule R1  Cost: 48.2962
 ^
 |
  <--- d31  Cost: 128
  <--- d51  Cost: 148
  <--- d42  Cost: 208

Total time = 532.296
Total Response time = 256.296

************************

Schedule R2  Cost: 60.832
 ^
 |
  <--- d64  Cost: 207
  <--- d34  Cost: 247

Total time = 514.832
Total Response time = 307.832

************************

Schedule R3  Cost: 33.722
 ^
 |
  <--- d51  Cost: 148
  <--- d12  Cost: 169
  <--- d11  Cost: 24.4839
 |      |  <--- d31  Cost: 128
 |      |  <--- d51  Cost: 148

Total time = 651.206
Total Response time = 206.206
```

```
************************
Schedule R4  Cost: 30.5804
 ^
 |
  <--- d12  Cost: 169
  <--- d13  Cost: 217
  <--- d62  Cost: 29.7912
 |      |  <--- d12  Cost: 169
 |      |  <--- d42  Cost: 208

Total time = 823.372
Total Response time = 268.372

************************

Schedule R5  Cost: 124.949
 ^
 |
  <--- d31  Cost: 128
  <--- d11  Cost: 24.4839
 |      |  <--- d31  Cost: 128
 |      |  <--- d51  Cost: 148

Total time = 553.433
Total Response time = 297.433

************************

Schedule R6  Cost: 27.2535
 ^
 |
  <--- d12  Cost: 169
  <--- d42  Cost: 208
  <--- d34  Cost: 247
  <--- d32  Cost: 20.1393
 |      |  <--- d12  Cost: 169
 |      |  <--- d42  Cost: 208
 |      |  <--- d62  Cost: 29.7912

Total time = 1078.18
Total Response time = 285.184
```

```
Statistical table for 6 relations with 3 attributes

 1200      0 0.000   256 0.200   115 0.116
 1600    198 0.177     0 0.000   151 0.152
 3700    319 0.285     0 0.000     0 0.000
 5200      0 0.000   418 0.327   312 0.312
 1800    224 0.200     0 0.000   194 0.195
 1600      0 0.000   276 0.216     0 0.000

AHY Algorithm (Response Time Version) Result

************************
Schedule R1  Cost: 31.0972
 ^
 |
 <--- d23  Cost: 171
 <--- d53  Cost: 23.4206
 |     | <--- d13  Cost: 135
 |     | <--- d23  Cost: 171
 <--- d43  Cost: 20.0189
 |     | <--- d13  Cost: 135
 |     | <--- d23  Cost: 171
 |     | <--- d53  Cost: 23.4206
 |           | <--- d13  Cost: 135
 |           | <--- d23  Cost: 171

Total time = 1186.96
Total Response time = 245.537

************************
Schedule R2  Cost: 31.2919
 ^
 |
 <--- d13  Cost: 135
 <--- d53  Cost: 23.4206
 |     | <--- d13  Cost: 135
 |     | <--- d23  Cost: 171
 <--- d43  Cost: 20.0189
 |     | <--- d13  Cost: 135
 |     | <--- d23  Cost: 171
 |     | <--- d53  Cost: 23.4206
 |           | <--- d13  Cost: 135
 |           | <--- d23  Cost: 171

Total time = 1151.15
Total Response time = 245.731

************************
Schedule R3  Cost: 150.98
 ^
 |
 <--- d21  Cost: 218
 <--- d51  Cost: 244

Total time = 612.98
Total Response time = 394.98

************************
Schedule R4  Cost: 37.8788
 ^
 |
 <--- d13  Cost: 135
 <--- d23  Cost: 171
 <--- d53  Cost: 23.4206
 |     | <--- d13  Cost: 135
 |     | <--- d23  Cost: 171

Total time = 673.299
Total Response time = 232.299

************************
Schedule R5  Cost: 51.7376
 ^
 |
 <--- d13  Cost: 135
 <--- d23  Cost: 171

Total time = 357.738
Total Response time = 222.738

************************
Schedule R6  Cost: 124.64
 ^
 |
 <--- d12  Cost: 276
 <--- d42  Cost: 38.0576
 |     | <--- d12  Cost: 276
 |     | <--- d62  Cost: 296

Total time = 1010.7
Total Response time = 458.698
```

Improvement Algorithms Result

```
************************
Schedule R1  Cost: 31.0972
 ^
 |
  <--- d23  Cost: 171
  <--- d53  Cost: 23.4206
 |      | <--- d13  Cost: 135
 |      | <--- d23  Cost: 171
  <--- d43  Cost: 20.0189
 |      | <--- d13  Cost: 135
 |      | <--- d23  Cost: 171
 |      | <--- d53  Cost: 23.4206

Total time = 880.957
Total Response time = 245.537

************************
Schedule R2  Cost: 31.2919
 ^
 |
  <--- d13  Cost: 135
  <--- d53  Cost: 23.4206
 |      | <--- d13  Cost: 135
 |
 |      | <--- d23  Cost: 171
  <--- d43  Cost: 20.0189
 |      | <--- d13  Cost: 135
 |      | <--- d23  Cost: 171
 |      | <--- d53  Cost: 23.4206

Total time = 845.152
Total Response time = 245.731

************************
Schedule R3  Cost: 150.98
 ^
 |
  <--- d21  Cost: 218
  <--- d51  Cost: 244

Total time = 612.98
Total Response time = 394.98
```

```
************************

Schedule R4  Cost: 37.8788
 ^
 |
  <--- d13  Cost: 135
  <--- d23  Cost: 171
  <--- d53  Cost: 23.4206
 |      | <--- d13  Cost: 135
 |
 |      | <--- d23  Cost: 171

Total time = 673.299
Total Response time = 232.299

************************

Schedule R5  Cost: 51.7376
 ^
 |
  <--- d13  Cost: 135
  <--- d23  Cost: 171

Total time = 357.738
Total Response time = 222.738

************************

Schedule R6  Cost: 124.64
 ^
 |
  <--- d12  Cost: 276
  <--- d42  Cost: 38.0576
 |      | <--- d12  Cost: 276
 |      | <--- d62  Cost: 296

Total time = 1010.7
Total Response time = 458.698
```

```
Statistical table for 4 relations with 3 attributes

  800    469 0.540000     0 0.000000     0 0.000000
  500      0 0.000000     0 0.000000   499 0.442000
 1600      0 0.000000   324 0.532000     0 0.000000
 2400    432 0.497000   270 0.444000   600 0.531000

AHY Algorithm (Response Time Version) Result
```

```
************************
Schedule R1  Cost: 417.6
 ^
 |
  <--- d41  Cost: 452

Total time = 869.6
Total Response time = 869.6

************************
Schedule R2  Cost: 285.5
 ^
 |
  <--- d43  Cost: 620

Total time = 905.5
Total Response time = 905.5

************************
Schedule R3  Cost: 730.4
 ^
 |
  <--- d42  Cost: 290

Total time = 1020.4
Total Response time = 1020.4

************************
Schedule R4  Cost: 324.747
 ^
 |
  <--- d32  Cost: 344
  <--- d11  Cost: 489
  <--- d23  Cost: 519

Total time = 1676.75
Total Response time = 843.747
```

```
Improvement Algorithms Result
```

```
************************
Schedule R1  Cost: 417.6
 ^
 |
  <--- d41  Cost: 452

Total time = 869.6
Total Response time = 869.6

************************

Schedule R2  Cost: 285.5
 ^
 |
  <--- d43  Cost: 620

Total time = 905.5
Total Response time = 905.5

************************
Schedule R3  Cost: 730.4
 ^
 |
  <--- d42  Cost: 290

Total time = 1020.4
Total Response time = 1020.4

************************
Schedule R4  Cost: 324.747
 ^
 |
  <--- d32  Cost: 344
  <--- d11  Cost: 489
  <--- d23  Cost: 519

Total time = 1676.75
Total Response time = 843.747
```

Bibliography

[AHY83] P. M.G. Apers, A. R. Hevner, S. B. Yao, Optimization Algorithms for Distributed Queries; IEEE 1983, 57-68

[BC81] P. A. Bernstein, and D.M. Chiu, Using semi-joins to solve relational queries; J. Ass. Comput. Mach., vol. 28 pp. 25-40, Jan. 1981

[BG81] P. A. Bernstein, and N. Goodman, The power of natural semi-join; SIAM J. Comput., vol. 10, no. 4, pp. 751-771, Nov 1981

[BGW+81]P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, J. B. Bothnie, Jr.,Query Processing in a System for Distributed Database (SDD-1); ACM Transactions on Database Systems, vol. 6, No. 4, December 1981, 602-625

[Blo70] B.H. Bloom, Space/time trade-offs in hash coding with allowable errors; Communications of the ACM, 13, 7, 422-426, 1970

[BR88] P. Bodorik, J.S. Riordon, Heuristic Algorithms for Distributed Query Processing; IEEE 1988, 144-155

[BRJ89] P. Bodorik, J.S. Riordon, C. Jacob, Dynamic Distributed Query Processing Techniques; ACM 1989, 348-357

[BRP92] P. Bodorik, J. S. Riordon, J. S. Pyra, Deciding to Correct Distributed Query Processing; IEEE Transactions on Knowledge and Data Engineering, vol. 4, No. 3, June 1992, 253-265

[CBTY89] A.L.P. Chen, D. Brill, M. Templeton, and C.T. Yu, Distributed query processing in a multiple database system; IEEE Journal on Selected Areas in Communications, 7(3):390-397, 1989

[CCY92] T.S. Chen, A. Chen, and W.P. Yang, Hash semijoin: A new technique for minimizing distributed query time; 3rd Workshop on Future Trends of Distributed Computing Systems, 325-330, 1992

[CG94] R.L. Cole, G. Graefe, Optimization of Dynamic Query Evaluation Plans; ACM 1994, 150-160

[CGP86] S. Ceri, G. Gottlob, and G. Pelagatti, Taxonomy and formal properties of distributed joins; Information Systems, 11, 1, 1986, 25-40

[Che80] T.Y. Cheung, A method for equi-join queries in distributed relational databases; IEEE Transaction on Computers, v31, 12, 1980

[CH80] D.M. Chiu, Y.C. Ho, A Methodology for Interpreting Tree Queries into Optimal Semi-Join Expression; ACM 1980, 169-178

[CI86] C.W. Chung and K.B. Irani, An optimization of queries in distributed database systems; Journal of Pallel and Distributed Computing, 3: 137-157, 1986

[CL84] A. L.P. Chen, V. O.K. Li, Improvement Algorithms for Semijoin Query Processing Programs in Distributed Database Systems; IEEE Transactions on Computers, vol. C-33, No.11, November 1984, 959-968

[CL98] H. Chen and C. Liu, Maintenance of Placement Dependency in Distributed Multidatabase System; ACM 1998

[CL84] J. S.J. Chen, V. O.K. Li, Improvement algorithms for semijoin query processing programs in distributed database systems; IEEE Transactions on Computing 33, Nov 1984

[CL90] J. S.J. Chen, V. O.K. Li, Domain-Specific Semijoins: A New Operation for Distributed Query Processing; Information Sciences 52, 165-183 (1990)

[CM95] P. Ciaccia and D. Maio, Domains and active domains: What this distinction implies for the estimation of projection sizes in relational databases; IEEE Transactions on Knowledge and Data Engineering, 641- 654, 1995

{CY88] D. Cornell, and P.S. Yu, Site assignment for relations and join operations in the distributed transaction processing environment; Conference on Data Engineering, 100-108, 1988

[CY90] M.S. Chen and P.S. Yu, Using join operations as reducers in distributed query processing; 2nd International Symp. On Databases in Parallel and Distributed System, 1990

[CY92] M.S. Chen and P.S. Yu, Interleaving a join sequence with semijoins in distributed query processing;IEEE Transactions on Parallel and Distributed Systems, 611-620, 1992

[CY93] M.S. Chen and P.S. Yu, Combining join and semijoin operations for distributed query processing; IEEE Transactions on Knowledge and Data Engineering, 534-542, 1993

[CY94] M.S. Chen and P.S. Yu, A graph theoretical approach to determine a join reducer sequence in distributed query processing; IEEE Transactions on Knowledge and Data Engineering, vol 6, 1, 152-165, 1994

[DHK97] T. Hoppler, A. Kemper, D. Kossmann, Database Performance in the Real World --- TPC-D and SAP R/3; J. Doppelhammer, ACM 1997, 123-134

[DL87] P.A. Dwyer and J.A. Larson, Some Experiences with a distributed database testbed, IEEE, 633-648, 1987

[Eps82] R. Epstein, Query Processing Techniques for Distributed Relational Database Systems; University Microfilms International, 1982

[Evr95] C. Evrendilek, Query optimization in multidatabase systems; Next Generation Information Technologies, 1995, 49-58

[ESW79] R. Epstein, M. Stonebraker, and E. Wong, Distributed Query Processing in a Relation Data Base System; ACM 1979, 169-180

[ES80] R. Epstein, and M. Stonebraker, Analysis of distributed database processing strategies; VLDB, 1980

[FK88] T. Furukawa and Y. Kambayashi, Efficient query processing methods for distributed databases with relational and network models; 2nd International Symposium on Interoperable Information Systems, 321-328, 1988

[FLM00] D. Florescu, A. Levy, I. Manolescu, D. Suciu, Query Optimization in the Presence of Limited Access Patterns; ACM 2000

[GHL+] Gropp, Huss-Lederman, Lumsdaine, Lusk, Nitzberg, Saphir, and Snir, MPI: The Complete Reference – 2nd Edition

[GLS] Gropp, Lusk, and Skjellum, Using MPI: Portable Parallel Programming with the Message-Passing Interface

[Gra93] G. Graefe, Query Evaluation Techniques for Large Databases; ACM computing surveys, vol 25, 2, June 1993

[GS82] N. Goodman and O. Shmueli, Tree queries: A simple class of relational queries; ACM Trans. on Database Sys., vol. 7, no. 4, pp. 653-677, Dec 1982

[GS86] B. Gavish and A. Segev, Set Query Optimization in Distributed Database Systems; ACM Transactions on Database Systems, 265-293, 1986

[GW89] G. Graefe, K. Ward, Dynamic Query Evaluation Plans; ACM 1989, 358-366

[HH00] P.J. Haas, J.M. Hellerstelin, Ripple Joins for Online Aggregation; ACM 2000

[HR93] E.R. Harris and L. Ramamohanarao, Optimal dynamic multi-attribute hashing for range queries; BIT, 561-579, 1993

[HY78] A.R. Hevner and A.B. Yao, Optimization of data access in distributed systems; Purdue University, Technical Report TR281, July 1978

[IFF98] Z.G. Ives, D. Florescu, M. Friedman, A. Levy, D.S. Weld, An Adaptive Query Execution System for Data Integration; ACM 1998

[Kos00] D. Kossmann, The State of the Art in Distributed Query Processing; ACM computing Surveys, vol. 32, No. 4, December 2000, 422-469

[Kro90] Z. Krolikowski, Evaluation and improvement of query processing methods in wide and local area network; In Proceedings of 5th Jerusalem Conference on Information Technology, 565-575, 1990

[KLK97] H. Kim, S. Lee, and H.J. Kim, Distributed query optimization using two-step pruning; Information and Software Technology, 39, 149-169, 1997

[KR87] H. Kang, N. Roussopoulos, Using 2-way Semijoins in Distributed Query Processing; IEEE 1987, 644-650

[KYY82] Y. Kambayashi, M. Yoshikawa, S. Yajima, Query Processing for Distributed Databases Using Generalized Semi-Joins; ACM 1982, 151-160

[Ma00] L. Ma, Query Optimization in Distributed DBMS; University of Windsor (Canada), Research Survey, 2000

[LC85] H. Lu, M. J. Carey, Some Experimental Results on Distributed Join Algorithms in a Local Network; Proc. of VLDB 85, Stockholm 292-304

[LL83] W. S. Luk, L. Luk, Optimizing semi-join programs for distributed query processing; 2nd Int. Conf. On Database, Aug. 1983 , pp. 298-316

[LOZ95] X. Lin, M.E. Orlowska, and X. Zhou, Using parallel semi-join reduction to minimize distributed query response time; 1st International Conference on Algorithms and Architectures for Parallel Processing, 517-526, 1995

[LPP91] P. Legato, G. Paletta, L.Palopoli, Optimization of Join Strategies in Distributed Databases; Information System vol. 16, No. 4, pp. 363-374, 1991

[LS91] A.Y. Lu and P.C. Sheu, Processing of multiple queries in distributed databases; 7th International Conference on Data Engineering, 42-49, 1991

[MOR01] J. Morrissey, Lecture note of Distributed Query Optimization (03-60-535), University of Windsor, 2001

[Mul90] J. K. Mullin, Optimal semijoins for distributed database systems; Information Systems vol. 16, No. 5, pp. 558-560, 1990

[Mul93] J. K. Mullin, Estimating the Size of a Relational Join; Information Systems vol. 18, No. 3, pp. 189-196, 1993

[MB95] J. Morrissey and S. Bandyopadhyay, Computer communication technology and its effect on distributed query optimization strategies; Canadian Conference on Electrical and Computer Engineering, 598-600, 1995

[MB97] J. Morrissey and W.T. Bealor, Minimizin data transfers in distributed query optimization: A comparative study and evaluation; Computer J., 39, 1997

[MBB95] J. Morrissey, S. Bandyopadhyay, and W.T. Bealor, A heuristic for minimizing total cost in distributed query processing; 7th International Conference on Computing and Information, 1995

[MBBK95]J. Morrissey, S. Bandyopadhyay, W.T. Bealor, and S. Kamat, Dynamic strategies and bloom filters for minimizing data transfers in distributed query optimization; 1995

[MBK96] J. Morrissey, W.T. Bealor, and S. Kamat, A comparitive evaluation of dynamic heuristics for cost minimization; 8th International Conference on Computing and Information, 1996

[ME92] P. Mishra, and M.H. Eich, Join Processing in Relational Database; ACM Computing Surveys, vol 24, no 1, March 1992

[MINH87] S. Masuyama, T. Ibaraki, S. Nishio, and T. Hasegawa, Shortest semijoin schedule for a local area distributed database system; IEEE Transaction Software Engineering, 13, 5, 1987, 602-606

[MM98] J. Morrissey and X. Ma, Investigating repose time minimization in distributed query optimization; ICCI, 1998

[MO97] J. Morrissey and W.K. Osborn, Experiments with the use of reduction filters in distributed query optimization; 9th IASTED International Conference on Parallel and Distributed Computing and Systems, 1997

[Mor96] J. Morrissey, Reduction filters for minimizing data transfers in distributed query optimization; Canadian Conference on Electrical and Computer Engineering, 1996

[MS88] K. Mikkillineni, and S.Y. Su, An evaluation of relational join algorithms in a pipelined query processing environment; IEEE Transaction Software Engineering, 14, 6, 1988, 838-848

[MT85] C. Meghini and C. Thanos, Querying fragmented relations in a distributed database, Acta Informatica, 22, 125-138, 1985

[NWMN99]K.W. Ng, Z. Wang, R.R. Muntz, S. Nittel, Dynamic Query Re-Optimization; Computer Science Department, University of California ACM 1999

[Ogu97] O. Ogunbadejo, Query Optimization in Distributed DBMSs: Horizontal Fragmentation in Distributed Systems; University of Windsor (Canada), Research Survey, 1997

[Osb97] W. Osborn, Methods for Distributed Query Optimization; University of Windsor (Canada), Research Survey, 1997

[OV99] M. T. Ozsu, P. Valduriez, Principles of Distributed Database Systems; Second Edition, Prentice Hall, Upper Saddle River, New Jersey 07458, 1999

[PC84] G. Piatetshy-Shapiro, and C. Connell, Accurate estimation of the number of tuples satisfying a condition; ACM SIGMOD Conference on the management of data, 1984

[PC90] W. Perrizo, C.S. Chen, Composite Semijoins in Distributed Query Processing; Elsevier Science Publishing Co., Inc. 1990, 196-217

[PER85] W. K. Perrizo, Upper bound response time semijoin strategies, in 1st International Conference on Supercomputer System, Dec. 1985, pp 273-279

[PLH89] W. Perrizo, J.Y.Y. Lin, and W. Hoffman, Algorithms for Distributed Query Processing in Broadcast Local Area Networks; IEEE Transaction Knowledge Data Engineering 1, 2, 215-225

[RG00] R. Ramakrishnan, J. Gehrke, Database Management System; Second Edition, McGraw-Hill,2000

[RK91] N. Roussopoulos, H. Kang, A Pipeline N-way Join Algorithm Based on the 2-way Semijoin Program; IEEE Transactions on Knowledge and Data Engineering, vol. 3, No. 4, December 1991, 486-494

[RRL97] C.C. Ribeito, C.D. Ribeito, and S.G. Lanzelottle, Query Optimization in distributed relational databases; Journal of Heuristics, 3, 5-23, 1997

[Seg91] A. Segev, Strategies for distributed query optimization; Information Sciences, 54, 67-88, 1991

[SAL96] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, A. Yu, Mariposa: A Wide-Area Distributed Database System; The VLDB Journal (1996) 5: 48-630

[SG88] A. Swami, A. Gupta, Optimization of large join queries; ACM 1988, 8-17

[SGC98] N. Shivakumar, H. Garcia-Molina, C.S. Chekuri, Filtering with Approximate Predicates; Proceedings of the 24th VLDB Conference, 1998, 263-274

[SHA98] S. Su, Y. Huang, and N. Akaboshi, Graph-Based Parallel Query Processing and Optimization Strategies for Object Oriented Database; Distributed and Parallel Database, 6, 247-258, 1998

[SS98] D. Subramanian, and K. Subramainian, Query Optimization in Multidatabase Systems; Distributed and Parallel Database 6, 183-210, 1998

[TC92] J. C.R. Tseng, A. L.P. Chen, Improving Distributed Query Processing by Hash-Semijoins; Journal of Info. Science and Engineering 8, 1992, 525-540

[TC94] P.S.M. Tsai and A.L.P. Chen, Optimizing entity join queries by extended semijoin in a wide area multidatabase environment. International Conference on Parallel and Distributed Systems, 676-681, 1994

[VG84] P. Valduriez, and G. Garderin, Join and semijoin algorithms for a multiprocessor database machine; ACM Transaction on database systems, 133-161, 1984

[VV84] Y. Varol, and S. Vrbsky, Distributed query processing allowing for redundant data; 4th International Conference on Distributed Computing Systems, 1984

[Wan96] C. Wang, The complexity of processing tree queries in distributed databases; 2nd IEEE Symposium on Parallel and Distributed Processing, 604-611, 1990

[Wc96] W.F. Wang and AL.P. Chen, A new relation partitioning scheme for distributed query optimization; Journal of Information Science and Engineering, 12, 79-99 1996

[WC96] C. Wang, and M.S. Chen, On the complexity of distributed query optimization; IEEE Tran. on Knowledge and Data Eng., vol 8, no 4, 1996

[WLC91] C. Wong, V. Li, and A. Chen, Distributed query optimization by one-shot fixed-precision semijoin execution; 7th International Conference on Data Engineering, 756-763, 1991

[Yao79] S.B. Yao, Optimization of query evaluation Algorithms; ACM Trans. Database System, 133-155, September 1979

[YC83] C.T. Yu, C.C. Chang, On the Design of a Query Processing Strategy in a Distributed Database Environment; ACM 1983, 30-39

[YC84] C.T. Yu, C.C. Chang, Distributed Query Processing; ACM Computing Survey, vol. 16, No. 4, December 1984, 399-433

[YL89] H. Yu, and S. Lafortune, An intelligent search method for query optimization by semijoin; IEEE Trans. Knowledge Data Engineering, 1, 2, 1989, 226-237

[ZL94] Q. Zhu and P.A. Larson, Establishing a fuzzy cost model for query optimization in a multidatabase system; 27th Annual Hawaii International Conference on System Sciences, 263-272, 1994

[ZL96] Q. Zhu and P.A. Larson, Building regression cost models for multidatabase systems; 4th International Conference on Parallel and Distributed Information Systems, 1996

VITA AUCTORIS

Name:	Nelson Chung Ngok, CHU
Place of Birth:	China
Year of Birth:	1975
Post-Secondary Education and Degree:	Hong Kong Technical College Hong Kong, China 1993-1996 Higher Diplomas in Electronic Engineering
	University of Windsor Windsor, Ontario, Canada 1998-2001 BSc. (Honours Computer Information System)
	University of Windsor Windsor, Ontario, Canada 2001-2002 MSc.
Honours and Awards:	China Tech Group Scholarship 1996
	University of Windsor In-Course Award 2001, 2002
	University of Windsor Graduate Scholarship 2002
Related Work Experience:	Sessional Instructor University of Windsor 2002
	Graduate / Teaching Assistant University of Windsor 2000-2002
	Owner and Manager Tristate Technology Company 1996-1997
	Customer Server Engineer Xerox (Hong Kong) Ltd. 1996-1997