Experiments with hash-semijoins in distributed query processing.

Olumuyiwa. Ogunbadejo

*University of Windsor*

Follow this and additional works at: [https://scholar.uwindsor.ca/etd](https://scholar.uwindsor.ca/etd)

**Recommended Citation**


[https://scholar.uwindsor.ca/etd/2032](https://scholar.uwindsor.ca/etd/2032)

This online database contains the full-text of PhD dissertations and Masters’ theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000 ext. 3208.
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®
NOTE TO USERS

Page(s) not included in the original manuscript are unavailable from the author or university. The manuscript was microfilmed as received.

viii

This reproduction is the best copy available

UMI
Experiments with Hash-Semijoins
in Distributed Query Processing

by
Olumuyiwa Ogunbadejo

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
1998
The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author’s permission.

L’auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L’auteur conserve la propriété du droit d’auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-52623-2
Abstract

Query processing in a distributed system requires the transmissions of data between computers in a network. For a given query, there exist several strategies that a distributed DBMS may choose to execute in order to generate results. All these strategies produce the same result but incur different query execution costs. The problem of finding an optimal query execution strategy in distributed database systems has been shown to be NP-hard [WC96]. Due to this, heuristic algorithms are necessary in solving the query processing problem but no matter what heuristics are used, the efficiency of executing a query depends heavily on the type of reducer used.

Employing the AHY Algorithm [AHY83] to decompose general queries into schedules of simple queries and using the hypothesis in the paper by Tseng and Chen [TC92], this thesis proposes two replacement algorithms; the first replaces all the semijoins in the schedules with hash-semijoins and the second replaces some of the semijoins in the schedule with hash-semijoins depending on which is cost-effective. Also, two heuristic algorithms are proposed based on the same heuristic as the AHY Algorithm (total Time); Algorithm AHY-H which generates schedules using hash-semijoin operator and Algorithm AHY-HS which generates schedules using either semijoin or hash-semijoin operator depending on which is cost-effective. This work also evaluates the performance of the proposed algorithms
of the proposed algorithms in comparison to the AHY Algorithm (total time). From our comparison study, we show that both Algorithm AHY-HS and AHY-H, efficiently reduce the cost of transmitting data in comparison to the AHY Algorithm and the hypothesis suggested in [TC92]. Also, there is no significant difference in performance between the two heuristic algorithm, Algorithm AHY-H and AHY-HS, using our experimental framework.
For my parents, Dr & Mrs T.A. Ogunbadejo
my sisters, Dele, Funke and Bussy
and my brother, Bayo.
Acknowledgments

I give thanks to Almighty God for giving me enough strength to complete this thesis successfully. No matter the level of intelligence of a student, it is always difficult to successfully complete an exercise of this nature without the assistance and guidance of good lecturers. In this case, I acknowledge the valuable and extensive assistance of my supervisor, Dr. J.M. Morrissey. This work could not have been possible without her kind of prompting and corrections. I would also thank both Dr. C. Ezeife and Dr. K. Hildebrandt for their comments and suggestions on my thesis.

My profound gratitude also goes to Wendy Osborn for her unequal help towards the acquisition of some of the information used in this thesis. Also, I appreciate the encouragements and comments from all my friends and fellow graduate students in my department.

Finally, I would like to acknowledge the financial and moral support from my family towards my academic pursuits.

Thank you all!
NOTE TO USERS

Page(s) not included in the original manuscript are unavailable from the author or university. The manuscript was microfilmed as received.

viii

This reproduction is the best copy available

UMI
# TABLE OF CONTENTS

Abstract ........................................ iv

Acknowledgments ................................... vii

List of Figures .................................... xii

List of Tables .................................... xiii

1 Introduction .................................... 1
   1.1 Distributed Query Processing .................. 1
   1.2 Query Optimization in DBMSs .................. 1
   1.3 Query Optimization Strategies ................. 2
   1.4 Phases in Distributed Query Processing ....... 4
   1.5 Processing Queries in Distributed DBMSs ...... 4
   1.6 Cost Model .................................. 6
   1.7 Problem Description .......................... 7
   1.8 Thesis Organization ........................... 9

2 Literature Review ............................. 10
   2.1 Notations and Definitions .................... 10
   2.2 Relational Operators used as Reducers ........ 13
      2.2.1 Semijoin Operator ....................... 13
2.2.2 Hash-Semijoin Operator ....................................... 17

2.2.2.1 Cost-effectiveness of the hash-semijoin operation
over the semijoin operation ....................................... 21

2.3 Semijoin-based Algorithms ................................. 22

2.4 Hash-semijoin based Algorithms ........................... 25

2.5 AHY (Apers-Hevner-Yao) Algorithms ..................... 26

2.5.1 Algorithm GENERAL ........................................ 27

2.5.2 Algorithm SERIAL ......................................... 29

2.5.3 Procedure TOTAL ......................................... 30

2.5.4 Complexity Analysis of Algorithm GENERAL (total
time) .......................................................... 31

3 Proposed Methodology .......................................... 33

3.1 Assumptions .................................................. 33

3.2 Proposed Algorithms ....................................... 34

3.2.1 Replacement Algorithms ................................. 34

3.2.1.1 Algorithm R1 ........................................... 34

3.2.1.2 Algorithm R2 ........................................... 35

3.2.2 Heuristic Algorithms ..................................... 37

3.2.2.1 Algorithm AHY-H ....................................... 38

3.2.2.2 Algorithm AHY-HS ..................................... 38
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>An Illustrative Example</td>
<td>39</td>
</tr>
<tr>
<td>3.4</td>
<td>Comparison of Results</td>
<td>59</td>
</tr>
<tr>
<td>4</td>
<td>Evaluation</td>
<td>61</td>
</tr>
<tr>
<td>4.1</td>
<td>Test Queries and Database</td>
<td>62</td>
</tr>
<tr>
<td>4.2</td>
<td>Experimental Results &amp; Discussions</td>
<td>63</td>
</tr>
<tr>
<td>4.2.1</td>
<td>Results from Experiment #1</td>
<td>66</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Results from Experiment #2</td>
<td>68</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Results from Experiment #3</td>
<td>70</td>
</tr>
<tr>
<td>4.3</td>
<td>Research Summary</td>
<td>71</td>
</tr>
<tr>
<td>5</td>
<td>Conclusions and Future Work</td>
<td>74</td>
</tr>
<tr>
<td>5.1</td>
<td>Overview</td>
<td>74</td>
</tr>
<tr>
<td>5.2</td>
<td>Conclusions</td>
<td>74</td>
</tr>
<tr>
<td>5.3</td>
<td>Recommendations for Future Research</td>
<td>76</td>
</tr>
<tr>
<td>Bibliography</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>Appendix A</td>
<td>Result Summaries</td>
<td>81</td>
</tr>
<tr>
<td>Appendix B</td>
<td>Vita Auctoris</td>
<td>85</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1.1  Query processing in a distributed database system  6
Figure 2.1  Sample of a schedule  ....................... 12
Figure 2.2  Illustration of how a semijoin operator reduces a
relation .................................................. 15
Figure 2.3  Illustration of how a hash-semijoin operator reduces
a relation (Adapted from [Ma97]) .................. 20
List of Tables

Table 3.1  Cost effectiveness Property .......................... 38
Table 3.2  Query statistics table ................................. 39
Table 4.1  Experimental Setup .................................. 64
Table 4.2  Descriptions of each of the experiment with the
           selectivity range ....................................... 64
Table 4.3  Average percentage improvement over the IFS
           using 0.02 – 0.2 selectivity ............................ 66
Table 4.4  Average percentage improvement over the IFS
           using 0.4 – 0.7 selectivity ............................. 68
Table 4.5  Average percentage improvement over the IFS
           using 0.7 – 1.0 selectivity ............................. 70
Table A.1  Average total cost using 0.02 – 0.2 selectivity ... 82
Table A.2  Average total cost using 0.4 – 0.7 selectivity .... 83
Table A.3  Average total cost using 0.7 – 1.0 selectivity .... 84
1.1 Distributed Query Processing

A distributed database is a collection of multiple, logically interrelated databases distributed over a computer network. In distributed processing, relations are often shipped between sites and the retrieval of data from those different distributed databases is referred to as distributed query processing. A database management system (DBMS) is a collection of integrated services which support database management and controls the creation, use and maintenance of a database. The DBMS provides a simple and unified interface to users so that they can access the databases as if there were a single database.

1.2 Query Optimization in DBMSs

In the last two decades, hundreds of papers have been published in the query optimization area and many distributed query optimization algorithms have been proposed [Ogu98]. These algorithms employ techniques such as optimal or heuristic searches\(^1\), join or semijoin operations, static or dynamic optimization, sequential or parallel execution and so on. A query optimization algorithm is an

---

\(^1\) To search for something by incremental exploration of an unknown terrain according to some criterion
algorithm that derives an optimal or close-to-optimal distribution strategy for a
given query.

A query optimization model consists of three components [Seg91]: an objec-
tive function (e.g. minimum transmission cost), parameters (e.g. size of relations
and join attributes), and constraints (e.g. “no fragmentation”, “local processing
done first”). The specification of these components reflects the user’s priorities,
the data in the database, and the hardware and software capabilities.

Formulating an optimal execution strategy for a query can only be accom-
plished by an exhaustive search of all possible strategies. The complexity of such
enumeration has been shown to be NP-hard [WC96], resulting in algorithms which
are computationally prohibitively expensive to use. Hence, heuristic algorithms
are employed to quickly formulate close-to-optimal strategies [AHY83, KR87].

1.3 Query Optimization Strategies

Most algorithms for determining query processing strategies in distributed
databases are static in nature. These algorithms are static in the sense that
once the sequence of operations is determined, they are executed without any
changes (irrespective of the circumstances). Such strategies rely heavily on
various techniques for estimating the sizes of partial results, selectivities and other
parameters pertaining to the distributed environment. The static optimization
strategies depend on accurate estimates of the sizes of intermediate results to
produce good semijoin schedules. If the estimations are inaccurate, the strategy may be far from optimal [ES80]. Two approaches have been introduced to solve this problem.

The first approach to solving this problem is to provide accurate estimates of the sizes of intermediate results so that good semijoin schedules can be produced but this information is often unattainable [BRJ89]. Attempts to derive accurate estimates are expensive in terms of the size and maintenance of the required statistical data. Also in [Bea95], it is shown that the estimated and actual cost of a schedule can be very different and this may lead to a schedule which is not optimal.

A second approach to solving this problem is to use a *dynamic* (adaptive) processing strategy [BRJ89]. Methods using this strategy have a high computational complexity which limits their applicability. The strategy can always be modified if it is not proceeding as planned, unlike the static processing strategy. Determining if it is proceeding as planned requires the gathering of information by one or more processors. The process of collecting information about a strategy is commonly referred to as *monitoring*. Based on this monitoring, there is a decision making process which decides whether the current strategy being executed should be aborted and a new strategy should be derived for the portion of the strategy that is yet to be processed. If the current strategy is to be aborted, then some form of *corrective* action will need to form and initiate a new strategy in
order to complete the query. Various methods of monitoring and corrective action are discussed in [BRJ89].

1.4 Phases in Distributed Query Processing

The processing of a query in a DBMS involves the transmission of data among different sites via a computer network. Since data are geographically distributed, the traditional approach to processing a distributed query involves the following three phases [AHY83]:

**Phase 1** *Local processing phase* - at each site involved in the query processing, all local processing such as selections and projections on the joining and target attributes is performed.

**Phase 2** *Reduction phase* - a sequence of semijoins is executed to reduce the size of relations in a cost-effective way to lessen the total communication cost.

**Phase 3** *Final processing phase* - all relations which are required to produce the result of the query are transmitted to the result site where the final query processing is performed.

1.5 Processing Queries in Distributed DBMSs

To process a query, a distributed database system contains four major subsystems [HY79]:
[i] The query processing subsystem: This subsystem consists of the query optimization analysis which checks for all the data needed to process a query and the optimization algorithm which designs an efficient distribution strategy for each query using the data materialization.

[ii] The integrity subsystem: This subsystem is responsible for solving the problem of controlling concurrent queries so that database integrity and consistency is maintained while the communication overhead of transmitting control information among network node is minimized.

[iii] The scheduler subsystem: This subsystem schedules the local processing and data transmissions of the strategy derived by the optimization algorithm. It coordinates the various schedules in the strategy so that the query response is presented at the result node.

[iv] The reliability subsystem: This subsystem monitors the system components continuously for failures. When a failure occurs, it distributes control information in the network so that normal operations can continue while the failure is being repaired.

In Figure 1.1, the interdependency between the subsystems is illustrated:
1.6 Cost Model

The cost model mirrors the costs encountered during the processing of the execution strategy. In processing a query, the costs incurred include the CPU, I/O and communication costs. The query optimization algorithm must be based on minimizing a particular cost/time function. Cost models are used to predict the execution cost of alternative execution plans for a query. Several cost functions have been proposed but most are simply adapted from previous research. Most
research is concerned with minimizing the cost (time) and/or response time incurred when processing and answering queries.

The minimum cost objective can be in the form of minimizing processing cost, communication cost or total cost. The total cost model objective is to minimize the overall cost incurred when processing a query while the response time model seeks to minimize the elapsed time to execute a query. In most approaches, the total cost is assumed to be the amount of data transferred.

The transmission cost between any two nodes in a distributed database system is defined as a linear function

\[ C(X) = C_0 + C_1 X \]

where \( X \) is the amount of data transmitted. In this thesis, we will define our cost measure in units of amount of data transmitted. The constant \( C_0 \) represents an initial start-up cost for each separate transmission and \( C_1 \) is a proportionality constant.

1.7 Problem Description

With the increasing demand for more and faster access to information that is geographically dispersed, various techniques to retrieve information efficiently have been proposed. Performance of the DDBS is critically dependent upon the ability of the query optimization algorithm to derive efficient query processing
strategies [ES80]. For a given query, there exist several strategies that a distributed DBMSs may choose to execute in order to generate results. All these strategies produce the same result but incur different query execution costs.

The principal problem in evaluating a relational query on a distributed database system is that relations must be joined and these relations may reside at different sites. For the join to be performed, one relation must be shipped to the other site. Data transmission is the dominant cost in such systems, so minimizing the amount of data to be shipped between nodes (sites) is of prime importance. If the data to be transmitted between nodes is moderately large, then significant queuing delays, synchronization delays and poor response time may result [AHY83]. To solve this problem, minimizing the amount of data being transferred will improve the response time and likewise reduce the total cost of processing the query. Due to this, the core of the optimization is the reduction phase and the primary concern is to generate a semijoin program that will be used in this phase.

The semijoin operation has traditionally been relied upon to reduce the cost of data transmission for distributed query processing. However, judiciously applying hash-semijoins as reducers can lead to a further reduction in the cost of transmitting data [TC92]. In view of this fact, we extend the work in [TC92] by exploring the hypothesis which states that “any semijoin program can be made more cost effective by replacing some or all the traditional semijoin operators with hash-semijoins”. We use the AHY algorithm to test the hypothesis. We also
propose two different algorithms for optimizing general queries: the first uses hash-semijoins to derive schedules and the second uses both hash-semijoins and semijoins to derive schedules.

1.8 Thesis Organization

This thesis is organized into five chapters. Chapter 1 introduced the problem area and provided relevant background material to this thesis. The introduction discussed the goal of distributed query processing, query optimization and processing strategies in DBMSs and cost models.

In chapter 2, a detailed literature review is presented. The review includes all the notations and definitions that are used throughout this thesis. A discussion of relational operators, semijoin and hash-semijoin algorithms is presented. A detailed description of the AHY algorithms is also presented.

Chapter 3 presents a detailed description of the proposed algorithms. An example comparing the proposed algorithms and the AHY algorithm is provided to clarify how each heuristic is executed.

In chapter 4, an evaluation of the proposed methodology is discussed. The remainder of this chapter is used to present the experimental results along with a discussion of the conclusions that can be inferred from the results.

Lastly, chapter 5 provides a summary of the conclusions obtained along with some suggestions for future work.
Chapter 2

Literature Review

This chapter will focus on the semijoin and hash-semijoin operators. The fundamental idea behind these two operators will be discussed, followed by an examination of applications based on these operators in distributed query processing.

In section 1, all the definitions and notations that are used throughout this thesis are presented. In section 2, the use of semijoins and hash-semijoins is explored. In section 3, a brief review of the algorithms based on semijoins is presented. In section 4, a brief review of the algorithms based on hash-semijoins is presented. In section 5, a detailed review of the AHY Algorithms is presented.

2.1 Notations and Definitions

The following notations are used in this thesis:

For each relation $R_i, i = 1, 2, ..., m$:

$A_i$ :- is number of distinct attributes in relation $R_i$ where $A_i > 1$.

$S(R_i)$ :- is the size (in bytes or any suitable measure) of $R_i$.

$|R_i|$ :- is the cardinality of relation $R_i$. 
For each attribute, \( d_{ij}, j = 1, 2, \ldots, A_i \) (where \( d_{ij} \) refers to the \( j \) join-attribute of relation \( R_i \)) of \( R_i \) the following are defined:

\[
D(d_{ij}) \ := \ \text{the domain of possible values for attribute \( d_{ij} \).}
\]

\[
|D(d_{ij})| \ := \ \text{the cardinality of} \ D(d_{ij}), \ \text{that is the number of distinct values that make up the domain for} \ d_{ij}.
\]

\[
|d_{ij}| \ := \ \text{the cardinality of relation} \ R_i \ \text{projected over attribute} \ d_{ij}, \ \text{that is the number of distinct values in attribute} \ d_{ij}.
\]

\[
S(d_{ij}) \ := \ \text{the size of attribute} \ d_{ij}.
\]

\[
S_f(d_{ij}) \ := \ \text{the size of Bloom filter for attribute} \ d_{ij}.
\]

\[
\rho(d_{ij}) \ := \ \text{the selectivity of attribute} \ d_{ij}, \ \text{where selectivity is defined as}
\]

\[
\frac{|d_{ij}|}{|D(d_{ij})|}
\]

**Definition 1.1:** **Schedule:** The data transmission used for reducing a relation and the transmission of the reduced relation to the query site form a schedule for a relation. An example of a schedule for a relation \( R_i \) can be seen in figure 2.1.
Attribute $d_{21}$ is sent to attribute $d_{31}$. A semijoin is performed on relation $R_3$. The reduced $d_{31}$ attribute is sent to relation $R_1$ in parallel with attribute $d_{22}$. Further relational operations reduce the size of relation $R_1$. Finally, the reduced relation $R_1$ is sent to the result node.

**Definition 1.2:** A *distribution strategy* for a query consists of the schedules for all relations which do not reside in the result node and are used in the query.

**Definition 1.3:** A *semijoin program* is a sequence of semijoins generated by the query optimizer. It can be viewed as a directed graph [YC84], where the vertices are the relations and a directed edge from one relation $R_i$ to another relation $R_j$; that is, $R_i \rightarrow R_j$ denotes the semijoin from $R_i$ to $R_j$ or as a schedule.

**Definition 1.4:** The *incoming selectivity* of a schedule for a relation (or attribute) is the product of selectivities of all the attributes in the schedule. More than one
occurrence of an attribute in a schedule can contribute only one instance of its selectivity.

**Definition 1.5:** A *general query* is one where each relation may contain more than one joining attribute. Such relations can be reduced in size by semijoins on the different joining attributes.

**Definition 1.6:** By *local processing*, we mean the computation of selection, projection and join operations between relations that reside in the same node.

### 2.2 Relational Operators used as Reducers

Various taxonomies exist for classifying distributed query processing algorithms. One method is to classify the algorithms on the type of operators used as reducers during the optimization phase of query processing.

Research has concentrated on query processing strategies which employ different relational operators such as joins [CY90], semijoins [AHY83, BGW+81, CGTW89, CL90, KR87, PC90, RK91, WLC93], or a combination of both [CY93].

In [TC92], a new relational operator, called a hash-semijoin, was proposed. This achieves the same reduction effects as the semijoin but at a lower cost.

#### 2.2.1 Semijoin Operator

The processing of queries requires shipping relations between sites in distributed relational databases [CP84]. The semijoin is an effective relational oper-
ator which was introduced to reduce the data transmission cost. A semijoin from relation \( R_j \) to relation \( R_i \), denoted by \( R_i \bowtie R_j \), is defined as \( \prod_{R_j} (R_i \bowtie R_j) \), where \( R_i \bowtie R_j \) is the join of \( R_i \) and \( R_j \), and \( \prod_{\lambda}(B) \) is the projection of relation \( B \) on the attributes of relation \( A \).

The diagrams below illustrates how the semijoin can be used to reduce the size of a relation. Given two relations \( R_i \) and \( R_j \) with join attribute \( A \), the semijoin \( R_i \bowtie R_j \) is computed as

\[
\begin{array}{c|c|c}
  R_j & \mathbf{B} \\
  \hline
  2 & 2 \\
  2 & 1 \\
  4 & 3 \\
  4 & 4 \\
  4 & 5 \\
  6 & 6 \\
\end{array}
\quad
\begin{array}{c|c|c}
  R_i & \mathbf{C} \\
  \hline
  1 & 2 \\
  2 & 3 \\
  3 & 6 \\
  4 & 8 \\
  5 & 7 \\
  6 & 9 \\
  7 & 5 \\
  7 & 4 \\
\end{array}
\]

The projection of relation \( R_j \) over the join attribute is:

\[
\begin{array}{c|c}
  R_j & \mathbf{A} \\
  \hline
  2 \\
  4 \\
  6 \\
\end{array}
\]

The reduced relation \( R'_j \) after sending the projection \( R'_j \) to the site of \( R_i \) is
As pointed out in [YC84, AHY83], the approach of using a semijoin operator as a reducer requires its implementation at the reduction phase where a sequence of semijoins is used to reduce the size of relations and to lessen the total communication cost required.

The cost associated with a semijoin refers to the cost of projecting the joining attribute from the reducing relation and transmitting the projected attribute to the relation to be reduced. The cost of performing the semijoin operation $d_{kj} \bowtie d_{ij}$ is estimated as

$$C(d_{kj} \bowtie d_{ij}) = C_0 + S(d_{ij}) \times C_1$$

where $C_0$ is the start-up cost and $C_1$ is some positive constant. All of the examples and the evaluation methodology used in this thesis assume that $C_0 = 0$ and $C_1 = 1$.

The semijoin method is beneficial if the cost to produce and send the projected relation to the other site is less than the cost of sending the whole operand relation and of doing the actual join (amount of size reduction on $d_{kj}$). The benefit of
performing the semijoin operation \( d_{kj} \bowtie d_{ij} \), denoted as \( B(d_{kj} \bowtie d_{ij}) \), is estimated as:

\[
S(R_k) - (S(R_k) \times \rho(d_{ij}))
\]

\[
\Leftrightarrow S(R_k)(1 - \rho(d_{ij}))
\]

where \( \rho(d_{ij}) \) is the selectivity of the attribute \( d_{ij} \).

The **net benefit** of a semijoin operation denoted as \( P(d_{kj} \bowtie d_{ij}) \), is defined as:

\[
B(d_{kj} \bowtie d_{ij}) - C(d_{kj} \bowtie d_{ij})
\]

A semijoin operation is said to be **profitable** if the net benefit is greater than zero \( (P(d_{kj} \bowtie d_{ij}) > 0) \).

The semijoin operation may be viewed as a filtering operation which eliminates tuples not needed for processing a query. Semijoins are guaranteed to monotonically reduce the size of a relation. In the worst case, the size of the resulting relation will be equal to the size of the relation being reduced. The semijoin is an asymmetric\(^2\) operator, so both applications of the semijoin must be considered to find the application with the greatest reduction. The semijoin approach is better if the semijoin acts as a sufficient reducer, that is, if only a few tuples of relation \( R \) participate in the join. Using semijoins to reduce data transmission costs may not be beneficial if the semijoin selectivities\(^3\) are high. Also, the semijoin approach is not beneficial in cases where all the tuples of relation

\(^2\) The semijoin \( R_i \bowtie R_j \) is not equivalent to the semijoin \( R_j \bowtie R_i \).

\(^3\) Selectivity is the number of different values occurring in the attribute divided by the number of all the possible values of the attribute.
$R$ participate in the join. It is better to send the entire relation to the query site because the additional transfer of a projection is required in the semijoin approach which increases the local processing cost.

### 2.2.2 Hash-Semijoin Operator

Hash-semijoins are constructed using search (Bloom) filters. A **Bloom filter** is an array of bits which functions as a very compact representation of the values of the join attributes. The idea of a Bloom filter was introduced by Bloom in [Blo70]. The concept of a Bloom filter is not new in database research; its first use in a database system is by Babb [Bab79] in the CAFS (Content Addressed File Store) device, where it was called a bit array store. The Bloom filter uses hashing techniques which are known to be efficient ways of finding matching values. The Bloom filter consists of a bit vector and hash transformations on the join attribute of interest. The bit vector filtering technique is used as a hashed semijoin to filter out tuples that have no matching tuples in a join [Mul90].

As summarized from [Mul90], the Bloom filter presents a probabilistic way of determining if an element is a member of a given set. It consists of a bit vector with $N$ bits and $t$ independent hash transformations whose range is the $N$ bits. Each element is hashed with each of the $t$ transforms and the addressed bits are set to one. To test membership, the element is hashed with each of the $t$ transforms and the resulting bit address are tested. If all of the addressed bits
are set, the element is accepted as a member of the set. If even one bit is not set, then that element could not possibly be a member of the set and it is rejected. Studies on Bloom filters in databases can be found in [QI88, SL76, Mul90]

The hash-semijoin of \( R_i \) and \( R_j \), denoted by \( R_j \prec R_i \) is computed as follows [TC92]:

a. Construct an bit array (Bloom filter) representing the semijoin projection of \( R_i \) and set all the bits to zero.

b. Using a hash function, produce an address for each of the join attributes.

c. For each address produced, set the corresponding bit in the Bloom filter to one.

d. Transmit the bit array to the site of \( R_j \).

e. For each tuple of \( R_j \), use the same hash function to hash the join attribute value to the bit array. If the join attribute of \( R_j \) is hashed to an address in the bit array that contains a one, the tuple of relation \( R_j \) is output at the result relation, else discard the tuple.

An example of how a hash-semijoin can be used to reduce the size of a relation is illustrated in figure 2.2. Given two relations \( R_i \) and \( R_j \) with join attribute \( A \), the hash-semijoin \( R_i \prec R_j \) is computed as follows:

- Create a Bloom filter and set all the bits to zero.

- Using the join attribute in \( R_j \), hash each value of the join attribute to an address in the Bloom filter. For each address produced, set the corresponding
bit in the Bloom filter to 1.

<table>
<thead>
<tr>
<th>Rj</th>
<th>A</th>
<th>B</th>
<th>h(value)</th>
<th>address</th>
<th>filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

- Send the filter to the site of relation R_i. Using the same hash function that was previously used, hash each of the join attribute value of relation R_i to the Bloom filter. If there is a hit (hashed to an address in the Bloom filter that contains a one), retain that tuple, else discard that tuple.

<table>
<thead>
<tr>
<th>Ri</th>
<th>A</th>
<th>C</th>
<th>h(value)</th>
<th>filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>h(1)→1, =&gt; reject</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>h(2)→2, =&gt; accept</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>3</td>
<td>h(3)→3, =&gt; reject</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>4</td>
<td>h(4)→4, =&gt; accept</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
<td>5</td>
<td>h(5)→5, =&gt; reject</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>6</td>
<td>h(6)→6, =&gt; accept</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>7</td>
<td>h(7)→7, =&gt; reject</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>7</td>
<td>h(7)→7, =&gt; reject</td>
<td>0</td>
</tr>
</tbody>
</table>
The reduced relation $R'_i$ after sending the Bloom filter and processed is

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.3 Illustration of how a hash-semijoin operator reduces a relation (Adapted from [Ma97])

The hash-semijoin achieves the same reduction as the semijoin but at a lower cost. The advantage of using the hash-semijoins over semijoins is that the space occupied by Bloom filters is relatively small compared to transmitting the actual join attribute itself. The disadvantage is that collisions between addresses may occur and can be resolved by increasing the size of the Bloom filter and carefully selecting a hash function [Mul83] or using multiple small Bloom filters. The use of multiple Bloom filters has an advantage of efficient usage of main memory [Mul90].

The cost associated with a hash-semijoin refers to the cost of creating and sending the Bloom filter from the reducing relation to the relation to be reduced. The cost of performing the hash-semijoin $d_{kj} \propto d_{ij}$ is estimated as

$$C(d_{kj} \propto d_{ij}) = C_0 + S_f(d_{ij}) \times C_1$$

where $C_0$ is the start-up cost and $C_1$ is some positive constant. In this thesis, a perfect hash function is used so that there are no false drops\(^4\).

\(^4\) Occurrence of collisions while constructing the Bloom filter
The hash-semijoin method is **beneficial** if the cost to perform the hash-semijoin operation is less than the cost of sending the whole operand relation. The benefit of performing the hash-semijoin operation $d_{kj} \propto d_{ij}$ is the amount of size reduction of relation $R_k$. Since the hash semijoin operator achieves the same reduction as the semijoin operator, the benefits are the same. Therefore $B(d_{kj} \propto d_{ij}) = B(d_{kj} \propto d_{ij})$.

The **net benefit** of a hash-semijoin operation is defined as

$$P(d_{kj} \propto d_{ij}) = B(d_{kj} \propto d_{ij}) - C(d_{kj} \propto d_{ij})$$

The hash-semijoin operation is said to be **profitable** if the net benefit is greater than zero ($P(d_{kj} \propto d_{ij}) > 0$).

**2.2.2.1 Cost-effectiveness of the hash-semijoin operation over the semijoin operation**

The hash-semijoin operation $d_{kj} \propto d_{ij}$ is said to be more cost-effective than the semijoin operation $d_{kj} \propto d_{ij}$, if the following condition holds:

$$P(d_{kj} \propto d_{ij}) - P(d_{kj} \propto d_{ij}) > 0$$

$$\Leftrightarrow C(d_{kj} \propto d_{ij}) - C(d_{kj} \propto d_{ij}) > 0$$

Since we assume that $C_0$ and $C_1$ are initialized to 0 and 1 respectively in this thesis, then this condition also holds

$$S(d_{ij}) - S_f(d_{ij}) > 0$$
where $S(d_{ij})$ is the size of the projected attribute $d_{ij}$ and $S_f(d_{ij})$ is the size of the Bloom filter to be transmitted. Therefore, the hash-semijoin is more cost-effective than the semijoin if the size of the Bloom filter is less than the size of the projected attribute transmitted.

$$S_f(d_{ij}) < S(d_{ij})$$

2.3 Semijoin-based Algorithms

One of the first algorithm using semijoins for distributed query processing was developed by Wong [Won77]. It was used in SDD-1 and was further refined in [BGW'81]. The objective of this algorithm is to process a query with a minimum amount of intersite data transfers by recursively finding the lower cost strategies until no further cost improvements can be discovered. The SDD-1 algorithm is based on a hill-climbing strategy which tries to apply as many beneficial semijoins as possible. Since SDD-1 algorithm is an iterative hill-climbing algorithm, it always selects the most profitable reduction that is immediately at hand. The major disadvantage of this approach lies in its inability to backtrack and consider other execution strategies which might produce better solutions.

The traditional semijoin operator is adequate for query optimization but recent studies showed that improvements could be made through the use of specialized operators. An extended version of the traditional semijoin called a 2-way semijoin is proposed in [RK91] as a method of improving the cost-effectiveness of the
semijoin operation. The 2-way semijoin is an extended operator in the sense that it produces two semijoins from its operands. It is shown theoretically that if the semijoin $A \bowtie B$ is cost-effective then $B \bowtie A$ is also cost-effective. The major concern of this operator is to maximize the reduction capacity of a semijoin operation by not only performing forward reduction as the traditional semijoin but also performs backwards reduction in a cost-effective way.

Most algorithms focus on the reduction effects of individual attributes of relations without considering the relationship between various relations. Due to this, the possibility of reducing multiple attributes is overlooked. Perrizo and Chen [PC90] suggest a composite semijoin, where multiple attributes are transmitted from a relation as a reducer. The composite semijoin is a refinement of the traditional semijoin where the projection and transmission involve multiple columns.

In a fragmented database, there is restricted use of the semijoin (can only be employed in cases of relation-to-relation or relation-to-fragment) operation to prevent the elimination of contributive tuples. Chen and Li [CL90] propose a operation called a domain-specific semijoin to address certain problems specific to horizontally fragmented databases. This operation exploits the semantic information associated with the joining fragmented relations and uses it to reduce the size of fragments by eliminating noncontributive tuples. In addition, the domain-specific semijoin could be applied in a fragment-to-fragment fashion without loss
of contributive tuples.

Numerous algorithms have been developed to determine a semijoin program for optimal distributed query processing. It is shown in [WC96] that finding an optimal semijoin strategy for a given query is NP-hard, so most research concentrates on developing heuristic algorithms which find an efficient but sub-optimal solution. Most semijoin algorithms favor executing semijoins sequentially so that the reduction effect of a semijoin can be propagated to reduce the cost of the later semijoins.

Performance studies in [LY93] have shown that such semijoin processing strategies are sometimes inefficient for the following reasons:

i. Loss of parallelism: The sequential execution of semijoin excludes the possibility of parallel semijoin execution in a distributed system.

ii. Relation scan overhead: Before $R_i$ and $R_j$ is executed, $R_j$ has to be scanned in order to generate the semijoin projection. If $R_k$ and $R_j$ also appears in the sequential semijoin program, $R_j$ has to be scanned again, which increases the processing overhead.

iii. Exclusion of n-way semijoin optimization: The sequential execution of a semijoin excludes the possibility of performing multiple semijoins to the same relation simultaneously.

iv. Inaccurate semijoin reduction estimation: If the estimation is done before the
semijoin processing, the accuracy may be low because estimation errors may propagate and be magnified through the sequential execution of semijoins.

A new semijoin processing method, called a one-shot semijoin execution was proposed in [WLC91], to remedy the inefficiency of traditional semijoin processing strategies which favor sequential execution of semijoins. All the necessary semijoin projections of a relation are generated simultaneously and all applicable semijoins are executed simultaneously. This method increases parallelism by allowing parallel generation of all the semijoin projections, parallel transmission of all the semijoin projections and parallel execution of all the semijoins. The number of disk accesses is reduced because a relation is scanned only once to generate all necessary semijoin projections. They developed a polynomial-time algorithm based on the response-time model and proposed the use of hashing to process multiple semijoins.

2.4 Hash-semijoin based Algorithms

Tseng and Chen [TC92] proposed the use of hash-semijoins as a more cost-effective distributed query processing method. It is based on the concept of a search (Bloom) filter. They propose a backwards replacement algorithm which replaces some of the traditional semijoins in a semijoin program with hash-semijoins. The algorithm works only for a special class of queries called tree queries and they suggest future work to devise an algorithm for distributed general
queries. Their approach was theoretically evaluated and was found to be more cost effective than the traditional semijoin.

2.5 AHY (Apers-Hevner-Yao) Algorithms

The AHY Algorithms are a collection of algorithms for optimizing a special class of queries called simple queries\(^5\) [AHY83]. These algorithms, namely Algorithm SERIAL and Algorithm PARALLEL are used to minimize the total cost and response time respectively.

The AHY algorithms are based on the followings assumptions:

[1] A point-to-point network is used where each computer is considered to be a node in the network. Each of these nodes has processing capability, data storage capability and is capable of autonomous operation within the DBMS.

[2] Transmission costs are significantly greater than local processing costs.

[3] Communication line contention and queuing delays are not considered.


[5] Attribute values are independent. This permits the reduction of the selectivity and size of the join attribute only.

In each of the algorithms, semijoins are used to reduce the size of relations by deleting those tuples which do not contribute in the final join.

---

\(^5\) a simple query is one which contains only one join attribute after initial local processing
Literature Review

If a relation $R_i$ has a semijoin with attribute $d_{kl}$ on attribute $d_{ij}$, then the parameters of relation $R_i$ are changed in the following way:

$$ s_i \leftarrow s_i \ast p_{kl} $$

$$ p_{ij} \leftarrow p_{ij} \ast p_{kl} $$

$$ b_{ij} \leftarrow b_{ij} \ast p_{kl} $$

Attributes or relations are transmitted to reducing a relation and the transmissions of the reduced relation to the query site form a schedule for this relation. The transmission cost is assumed to be a linear function of the size of the data. The function is denoted by

$$ C(X) = C_0 + C_1X $$

where $C_0$ is a constant cost associated with initiating a data transfer, $C_1$ is a cost coefficient associated with the amount of data transmitted ($X$).

Since this thesis is based on finding the minimum total cost of processing a query, the discussion of the AHY Algorithm will be limited to Algorithm SERIAL and Algorithm GENERAL (total time).

2.5.1 Algorithm GENERAL

Algorithm GENERAL is a semijoin heuristic that uses an improved exhaustive search to find efficient distribution strategies. This is a straightforward extension of Algorithm SERIAL and PARALLEL which is used to optimize general queries.\[6\]

---

\[6\] A general query is a query with joins and unions in their optimization graph

University of Windsor, 1998
The overall strategy employed by Algorithm GENERAL is the decomposition of a general query into a collection of simple subqueries forming schedules. The resulting schedules produced are examined and integrated to form an optimization schedule representing the general query.

Two versions of the algorithm were developed depending on the cost function in use. To minimize the response time, parallel data transmission was emphasized by the use of Algorithm PARALLEL and Procedure RESPONSE in Algorithm GENERAL (response time). To minimize the total time, serial time transmission was emphasized by the use of Algorithm SERIAL and Procedure TOTAL in Algorithm GENERAL (total time). Also, a third version of the algorithm was developed that uses Algorithm SERIAL and Procedure COLLECTIVE in Algorithm GENERAL (collective), to produce strategies with increased data transmission redundancy among schedules.

A detailed outline of Algorithm GENERAL, summarized from [AHY83], is given below:

**Algorithm GENERAL**

i. *Do all initial local processing.*

ii. *Generate candidate schedules.* Isolate each of the \( \sigma \) joining attributes, and consider each to define a simple query with an undefined result node.

   a. To minimize *response time*, apply Algorithm PARALLEL to each simple
query. Save all candidate schedules for integration in step 3.

b. To minimize total time, apply Algorithm SERIAL to each simple query. This results in one schedule per simple query. From these schedules, the candidate schedules for each join are extracted. Consider the common join attribute $d_{ij}$. Its candidate schedule is identical to the schedule produced by algorithm SERIAL, applied to simple query in which $d_{ij}$ occurs, up to the transmission of $d_{ij}$. All transmissions after that are deleted from the schedule.

iii. Integrate the candidate schedules. For each relation $R_i$, the candidate schedules are integrated to form a processing schedule for $R_i$. If response time is to be minimized, procedure RESPONSE is used to integrate the schedules. Otherwise, to minimize the total time, schedule integration is performed by using either procedure TOTAL or procedure COLLECTIVE.

iv. Remove schedule redundancies. Eliminate relation schedules for relations which have been transmitted in the schedule for another relation.

### 2.5.2 Algorithm SERIAL

Algorithm SERIAL is used to reduce the total cost of processing a query by arranging the transmissions of relation in ascending order of size. An outline of Algorithm SERIAL, summarized from [AHY83], is given below:

**Algorithm SERIAL**
i. Order relations $R_i$ such that $s_1 \leq s_2 \leq \ldots \leq s_m$ and where $s_i$ is the size of relation $R_i$.

ii. If no relation are at the result node, then select strategy

$$R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_n \rightarrow \text{result node}$$

or else if $R_r$ is a relation at the result node, then there are two strategies:

$$R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_r \rightarrow \ldots \rightarrow R_n \rightarrow R_r$$

$$R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R_{r-1} \rightarrow R_{r+1} \rightarrow \ldots \rightarrow R_n \rightarrow R_r$$

2.5.3 Procedure TOTAL

Procedure TOTAL (summarized from [AHY83])

i. Adding candidate schedules. For each relation $R_i$ and each candidate schedule $CSCH_i$, perform the following. If a schedule contains a transmission of joining attribute of $R_i$, say $d_{ij}$, then add another candidate schedule identical to $CSCH_i$ except that the transmission of $d_{ij}$ is deleted.

ii. Select the best candidate schedule. For each relation $R_i$, and each joining attribute $d_{ij} (j = 1, 2, \ldots \sigma)$, select the candidate schedule which minimizes the total time for transmitting $R_i$. Only joining attributes which can be joined with $d_{ij}$ are considered. $BEST_{ij}$ denotes the best candidate schedule for relation $R_i$ and joining attribute $d_{ij}$.

iii. Candidate schedule ordering. For each relation $R_i$, candidate schedules $BEST_{ij}$ are ordered on joining attributes $d_{ij}, j = 1, 2, \ldots \sigma$, so that $ART_{i1} +$
\[ C(s_1 \ast SLT_{11}) \leq \ldots \leq ART_{i\sigma} + C(s_i \ast SLT_{i\sigma}). \] Schedules involving joining attributes not in \( R_i \) are disregarded. \( ART_{ij} \) denotes the arrival time of the \( BEST_{ij} \) schedule. \( SLT_{ij} \) denotes the accumulated selectivity of the \( BEST_{ij} \) schedule into \( R_i \).

iv. **Schedule integration.** For each \( BEST_{ij} \) in ascending order of \( j \), construct the integrated schedule to \( R_i \) which consists of the parallel transmissions of candidate schedules \( BEST_{ij} \) and \( BEST_{ik} \) where \( K < j \). The integrated schedule resulting in the minimum total time value is selected, where the total time value is computed by:

\[
TOTT_i = \sum_{k=1}^{j} \left[ ART_{ik} + C \left( s_i \ast \prod_{k=1}^{j} SLT_{ik} \right) \right]
\]

### 2.5.4 Complexity Analysis of Algorithm GENERAL (total time)

The following discussion on the complexity analysis of Algorithm GENERAL (total time) is summarized from [AHY83].

Assuming that a general query requires data from \( m \) relations and are joined on \( \sigma \) joining attributes. In step 2, Algorithm SERIAL is applied to each simple query. Since the joining attributes have to be ordered by size, the complexity is \( O(\sigma m \log_2 m) \).
The complexity of procedure Total is $O(\sigma m^2)$. In step 1, no more than $O(\sigma m)$ candidate schedules are added. This means that for each relation, the procedure must determine the $BEST_{ij}$ schedule among $O(\sigma m)$ candidate schedules. Hence, the complexity of step 2 is $O(\sigma m^2)$. Therefore, the complexity for an arbitrary general distributed query, Algorithm GENERAL (total time) has a processing complexity no worse than $O(\sigma m^2)$.
Chapter 3
Proposed Methodology

In this chapter, assumptions that are used throughout this thesis are discussed, the different methodologies behind the proposed algorithms are clearly stated and a comparative example is used to illustrate how each proposed algorithm is executed. Algorithm GENERAL (total time) is used as a benchmark for illustration.

Section 1 outlines all the assumptions used. In section 2, the methodology behind the proposed algorithms is discussed in detail. In section 3, an illustrative example for the proposed algorithms and Algorithm GENERAL (total time) is given. Finally, the conclusions drawn from the comparison of the illustrative example are stated.

3.1 Assumptions

In evaluating the algorithms, the framework used is based on the following assumption and objectives:

- Point-to-point network, select-project-join (SPJ) queries only and no data replication or fragmentation.
- Each relation in the query is at a different site and has one non-joining attribute which is required at the query site.
Proposed Methodology

- The cost of projecting and performing a hash function on the projection is considered to be negligible.
- A perfect hash-function is used which simply takes the attribute value as the address. This hashing function does not permit collisions.
- A 64 bit machine is assumed for the purpose of the evaluation in this thesis so that data are transmitted in a 64 bit word representation.
- Attribute values are independent.

3.2 Proposed Algorithms

In this section, a detailed description of each of the proposed algorithm is presented. The proposed algorithms consist of two replacement algorithms and two heuristic algorithms.

3.2.1 Replacement Algorithms

3.2.1.1 Algorithm R1

Algorithm R1 is a replacement algorithm which replaces all the semijoin operations in an execution schedule with hash-semijoin operations. This is done in a forward manner from node to node by replacing the semijoin operations with hash-semijoin operations. We will describe the algorithm with a running example.
Proposed Methodology

We have the schedule below and a domain size of 1000. The Bloom filter size

\[
\text{is calculated by dividing the domain size of the attribute by the bit representation of}
\text{the machine used. Therefore, the size of the Bloom filter is } \text{Ceil}(1000/64) = 16.
\]

The algorithm replaces the semijoin operations \((R_1 \bowtie d_{22}) \bowtie d_{31} \bowtie d_{21}\) with \((R_1 \bowtie d_{22}) \bowtie d_{31} \bowtie d_{21}\). The output from this algorithm is given below

3.2.1.2 Algorithm R2

Algorithm R2, which is also a replacement algorithm, is based on the hypothesis suggested by Tseng and Chen [TC92]. The algorithm transforms a semijoin program (schedule) into a cost-effective one by replacing some of the semijoin
operations with hash-semijoin operations. The replacement is done in a forward manner by checking from node to node to determine whether performing a semijoin operation or a hash-semijoin operation is cost-effective. The cost-effectiveness property is discussed in section 2.2.2.1. The algorithm is as follows

**Algorithm:** Forward replacement of some traditional semijoins with hash-semijoins

**Input:** An execution schedule representing how a relation is reduced before been transmitted to a query site.

**Output:** An execution schedule based on both hash-semijoin and semijoin operations.

**Begin**

**While** depth\(^7\) \(\neq 0\)

Given a sequence of semijoins \(d_{a_1} \times d_{b_1} \times \ldots \times d_{m_j} \times R_i\)\(^8\) representing a schedule for relation \(R_i\), set pointer to first reducer \(d_{a_1}\)

**While** pointer \(\neq R_i\)

**If** \(S_f(d_{a_1}) < S(d_{a_1})\)\(^9\) **Then**

Replace \(d_{b_j} \prec d_{a_1}\) with \(d_{b_j} \prec d_{a_j}\)

---

\(^7\) Number of parallel transmissions of different joining attributes in a schedule  
\(^8\) \(d_{a_1} \times d_{b_1} \times \ldots \times d_{m_j} \times R_i \equiv R_i, d_{m_j} \times \ldots \times d_{b_j} \times d_{a_j}\)  
\(^9\) Where \(S_f(d_{a_1})\): is the size of Bloom filter for attribute \(d_{a_1}\) and \(S(d_{a_1})\): is the size of attribute \(d_{a_1}\)
Else

Retain \( d_{bj} < d_{aj} \)

End-if

set pointer to successor node

End-while

decrement depth by one

End-while

End-of-algorithm

3.2.2 Heuristic Algorithms

Here we present two static heuristic algorithms, which aim to minimize the cost of transmitting data during distributed general query processing. Each of the proposed heuristic algorithms presented is based on the same heuristic as the AHY Algorithm (total time) but differs in the way reducers\(^{10}\) are transmitted to the reducing attribute/relation. The proposed algorithms decompose a general query into simple queries and generate schedules for each simple query. The resulting schedules are examined and integrated to generate an optimal or close-to-optimal execution strategy representing the general query.

\(^{10}\) Any attribute which can be used to eliminate tuples of an attribute/relation that will not contribute in deriving the final result

University of Windsor, 1998
3.2.2.1 Algorithm AHY-H

Algorithm AHY-H has the same principle of constructing the reducers as the AHY algorithm but instead of using semijoins, we use hash-semijoins so that Bloom filters are transmitted rather than the projection of the attributes. A running example to illustrate how this algorithm constructs execution schedules is given in section 3.3.

3.2.2.2 Algorithm AHY-HS

Algorithm AHY-HS is an improvement over Algorithm GENERAL (total time) in the sense that it attempts to construct reducers for each attribute in a cost effective manner. Each of the reducers is built by determining whether using a semijoin operation or a hash-semijoin operation is cost-effective. Given attributes $d_{aj}, d_{bj}, d_{cj}, ..., d_{mj}$ such that $S(R_d) \leq S(R_b) \leq S(R_c) \leq ... \leq S(R_m)$, this algorithm constructs reducers using the condition stated below

```
IF $S_f(d_{aj}) < S(d_{aj})$ THEN
    hash-semijoin operation $d_{bj} \bowtie d_{aj}$ is used as the reducer
ELSE
    semijoin operation $d_{bj} \bowtie d_{aj}$ is used
END-IF
```

Table 3.1 Cost effectiveness Property
A running example to illustrating how this algorithm constructs execution schedules is given in section 3.3.

3.3 An Illustrative Example

To illustrate and compare the proposed algorithms with Algorithm GENERAL (total cost), the execution of each algorithm on the statistical table given in Table 3.1 is outlined in detail.

In all of the examples and in the evaluation in this thesis, it is assumed that the cost is simply the size of the projected attribute when a semijoin is used or the size of the filter when a hash-semijoin is used.

<table>
<thead>
<tr>
<th>Relation</th>
<th>Size</th>
<th>( b_{i1} )</th>
<th>( d_{i1} )</th>
<th>( p_{i1} )</th>
<th>( b_{i2} )</th>
<th>( d_{i2} )</th>
<th>( p_{i2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 )</td>
<td>1700</td>
<td>170</td>
<td>0.15</td>
<td>164</td>
<td>0.19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( R_2 )</td>
<td>2640</td>
<td>125</td>
<td>0.11</td>
<td>26</td>
<td>0.03</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( R_3 )</td>
<td>2720</td>
<td>147</td>
<td>0.13</td>
<td>-</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.2 Query statistics table

Initial Feasible Solution

The easiest way to process a query submitted to a distributed database system is to ship all the relations involved to the result site for processing. The initial feasible solution is the cost of sending all the relations to the result site for
Proposed Methodology

processing. In this case, the initial feasible solution total cost is 7060. This strategy could be prohibitively expensive if the relations involved are very large.

\[
\begin{align*}
R_1 : & \quad 1700 \\
R_2 : & \quad 2640 \\
R_3 : & \quad 2720
\end{align*}
\]

**TOTAL COST = 7060**

**Applying Algorithm GENERAL (total time)**

After initial processing, two simple queries are formed on attributes \(d_{i1}\) and \(d_{i2}\). In step 2 of Algorithm GENERAL (total time), the following serial candidate schedules are formed.
For $d_{11}$:

Only the construction of the schedule for relation $R_1$ will be discussed in detail.

For attribute $d_{11}$,

In step 1 of procedure TOTAL, each candidate schedule for attribute $d_{11}$ is examined to determine if new schedules could be formed. The following two new schedules are added to the schedules for $d_{11}$:
These schedules are obtained by deleting the transmission of attribute $d_{11}$ from the schedules for $d_{21}$ and $d_{31}$ respectively. In step 2, for attribute $d_{11}$, the schedule $d_{11}$ is not considered since it cannot reduce itself. Only the remaining four schedules need to be tested.

Total time = $C(170) + C(0.15 \times 125) + C(0.11 \times 1700)$
= $170 + 19 + 187$
= 376

Total time = $C(125) + C(0.11 \times 1700)$
= $125 + 187$
= 312
Proposed Methodology

Total time = \( C(170) + C(0.15 \times 125) + C(0.15 \times 0.11 \times 147) \)
+ \( C(0.11 \times 0.13 \times 1700) \)
= 170 + 19 + 3 + 25
= 217

Total time = \( C(125) + C(0.15 \times 125) + C(0.11 \times 0.13 \times 1700) \)
= 125 + 17 + 25
= 167

Clearly, the schedule of \( d'_{31} \) has the minimum total time, hence it is chosen as the \( BEST_{11} \) schedule.

For attribute \( d_{12} \),

In step 1 of procedure TOTAL, only one schedule is added to the schedules for \( d_{12} \):

In step 2, schedule \( d_{12} \) is redundant to \( R_1 \), hence it is not considered.

University of Windsor, 1998
Proposed Methodology

\[ \text{Total time} = C(164) + C(0.19 \times 26) + C(0.03 \times 1700) \]
\[ = 164 + 5 + 51 \]
\[ = 220 \]

\[ \text{Total time} = C(26) + C(0.03 \times 1700) \]
\[ = 26 + 51 \]
\[ = 77 \]

On inspection, schedule \( d_{22}' \) has the minimum total time and is chosen as the \( \text{BEST}_{12} \) schedule.

In step 3 and 4, the \( \text{BEST}_{1j} \) schedules are ordered on their total time and the following two integrated schedules are constructed.

\[ \text{Total time} = C(26) + C(0.03 \times 1700) \]
\[ = 26 + 51 \]
\[ = 77 \]
The first of these two has the smallest total time, so it is chosen as the solution of Algorithm GENERAL for relation $R_1$.

The schedules for relations $R_2$ and $R_3$ are constructed in a similar way. The complete Algorithm GENERAL (total time) strategy for the given query is

Based on the statistical query table given, when Algorithm GENERAL (total time) was applied, it produced execution schedules with total cost of 555.
Applying Algorithm R1

To apply Algorithm R1 to the schedules produced by Algorithm GENERAL (total time), we simply replace all the semijoins in the schedule with hash-semijoins. The size of the Bloom filter is the domain size of the attribute divided by 64 (because we assume all operations are performed on a 64 bit machine). For attribute $d_{i1}$ and attribute $d_{i2}$, the size of the Bloom filter is $Ceil(1130/64) = 18$ and $Ceil(860/64) = 14$ respectively. Given the resulting schedules from Algorithm GENERAL (total time):

For each schedule of each relation, we replace semijoins with hash-semijoins so that Bloom filters are transmitted rather than the projected attributes. For relation $R_1$, the semijoin operation $R_1 \bowtie d_{22}$ is replaced with $R_1 \bowtie d_{22}$. This is done for the schedules of $R_2$ and $R_3$, and the resulting solution is given below:
Proposed Methodology

\[ \text{Total time} = C(14) + C(0.03 \times 1700) \]
\[ = 14 + 51 \]
\[ = 65 \]

\[ \text{Total time} = C(18) + C(18) + C(18) + C(0.11 \times 0.13 \times 2640) \]
\[ = 18 + 18 + 18 + 52 \]
\[ = 106 \]

\[ \text{Total time} = C(18) + C(18) + C(0.15 \times 0.11 \times 2720) \]
\[ = 18 + 18 + 45 \]
\[ = 81 \]

The total cost using Algorithm R1 is 252.

**Applying Algorithm R2**

This is similar to Algorithm R1. It replaces some of the semijoins in a schedule with hash-semijoins depending on whether or not it is cost-effective.

Also, applying Algorithm R2 to the schedules produced by Algorithm GENERAL (total time):
For relation $R_1$,

the cost of the semijoin operation $C(R_1 \bowtie d_{22}) = 26$ and the cost of the hash-
semijoin operation $C(R_1 \bowtie d_{22}) = 14$. Since the cost of the hash-semijoin is
less than the semijoin operation, the semijoin operation is replaced with a hash-
semijoin operation.

For relation $R_2$,

$C(d_{21} \bowtie d_{11}) = 170 > C(d_{21} \bowtie d_{11}) = 18$ so $d_{21} \bowtie d_{11}$ is replaced with $d_{21} \bowtie d_{11}$.

$C(d_{31} \bowtie (d_{21} \bowtie d_{11})) = 19 > C(d_{31} \bowtie (d_{21} \bowtie d_{11})) = 18$, therefore it is also
replaced with the hash-semijoin operator. Finally, $C(R_2 \bowtie (d_{31} \bowtie d_{21} \bowtie d_{11})) = 3$ is less than $C(R_2 \bowtie (d_{31} \bowtie d_{21} \bowtie d_{11})) = 18$ and therefore the semijoin
operation is retained. This is repeated for relation $R_3$ and the result is given
below:
The total cost using Algorithm R1 is 237.

**Applying Algorithm AHY-H**

In this case, hash-semijoin operators are used as the reducer so that Bloom filters are sent rather than the projected attribute. We generate serial candidate schedules for attributes $d_{i1}$ and $d_{i2}$ using filters.
Similar to the previous example (Algorithm GENERAL), only the construction of the schedule for $R_1$ will be discussed in detail. For attribute $d_{11}$, examine candidate schedules to see if new schedules could be formed. Eliminate schedule $d_{11}$ since it is redundant to $R_1$. Repeat the process for attribute $d_{12}$.

For attribute $d_{i1}$, the size of the filter is 18 while for attribute $d_{i2}$ the size of the filter is 14.
Proposed Methodology

Total time = \( C(18) + C(18) + C(0.11 \times 1700) \)
= \( 18 + 18 + 187 \)
= 223

Total time = \( C(19) + C(0.11 \times 1700) \)
= \( 18 + 187 \)
= 205

Total time = \( C(18) + C(18) + C(18) + C(0.11 \times 0.13 \times 1700) \)
= \( 18 + 18 + 18 + 25 \)
= 79

Total time = \( C(18) + C(18) + C(0.11 \times 0.13 \times 1700) \)
= \( 18 + 18 + 25 \)
= 61

Total time = \( C(14) + C(14) + C(0.03 \times 1700) \)
= \( 14 + 14 + 51 \)
= 79
Proposed Methodology

\[ \text{Total time} = C(14) + C(0.03 \times 1700) \]
\[ = 14 + 51 \]
\[ = 65 \]

We calculate the cost of each schedule for attribute \( d_{11} \) and choose the best schedule. The best schedule \( BEST_{11} \) is schedule \( d'_{31} \). We repeat the process for attribute \( d_{12} \), the best schedule \( BEST_{12} \) is schedule \( d'_{22} \). Order the \( BEST_{ij} \) schedules on the total cost and integrate the schedules.

\[ \text{Total time} = C(18) + C(18) + C(0.11 \times 0.13 \times 1700) \]
\[ = 18 + 18 + 25 \]
\[ = 61 \]

\[ \text{Total time} = C(18) + C(18) + C(14) + C(0.11 \times 0.13 \times 0.03 \times 1700) \]
\[ = 18 + 18 + 14 + 1 \]
\[ = 51 \]

Choose the smallest of the integrated schedules as the solution of Algorithm AHY-H for relation \( R_1 \).

University of Windsor, 1998
The schedules for relations $R_2$ and $R_3$ are constructed in a similar way. The complete Algorithm AHY-H strategy derived for the given query is

Based on the statistical query table given, when Algorithm AHY-H was applied, it produced execution schedules with a total cost of 192.

**Applying Algorithm AHY-HS**

This is similar to Algorithm GENERAL (total time), except that it checks to determine whether performing a semijoin or hash-semijoin operation is more cost-effective when creating schedules.

University of Windsor, 1998
Also, we generate serial candidate schedules for attributes $d_{11}$ and $d_{12}$.

Only the construction of the schedule for $R_1$ will be discussed in detail. For attribute $d_{11}$, examine candidate schedules to see if new schedules could be formed. Eliminate schedule $d_{11}$ since it is redundant to $R_1$. Repeat the process for attribute $d_{12}$.
For each schedule, the amount of data transferred between sites using the semijoin or hash-semijoin operation is calculated and whichever is less is adopted. For attribute \( d_{i1} \), the size of the filter is 18, while for attribute \( d_{i2} \), the size of the filter is 14.

For schedule \( d_{21} \), \( C(d_{21} \bowtie d_{11}) = 170 \) and \( C(d_{21} \bowtie d_{11}) = 18 \) so the hash-semijoin operation is used. \( C(R_1 \bowtie (d_{21} \bowtie d_{11})) = 19 \) and \( C(R_1 \bowtie (d_{21} \bowtie d_{11})) = 18 \) so the hash-semijoin operation is also used.

For schedule \( d'_{21} \), \( C(R_1 \bowtie d_{21}) = 125 \) and \( C(R_1 \bowtie d_{21}) = 18 \), so the hash-semijoin operation is used.

For schedule \( d_{31} \), \( C(d_{31} \bowtie d_{11}) = 170 \) and \( C(d_{31} \bowtie d_{11}) = 18 \), so the hash-semijoin operation is used. \( C(d_{31} \bowtie (d_{31} \bowtie d_{11})) = 19 \) and \( C(d_{31} \bowtie (d_{21} \bowtie d_{11})) = 18 \), so the hash-semijoin operation is also used. \( C(R_1 \bowtie (d_{31} \bowtie (d_{31} \bowtie d_{11}))) = 3 \) and \( C(R_1 \bowtie (d_{31} \bowtie (d_{21} \bowtie d_{11}))) = 18 \), so the semijoin operation is used.

For schedule \( d'_{31} \), \( C(d_{31} \bowtie d_{21}) = 125 \) and \( C(d_{31} \bowtie d_{21}) = 18 \), so the hash-semijoin operation is used. \( C(R_1 \bowtie (d_{31} \bowtie d_{21})) = 17 \) and \( C(R_1 \bowtie (d_{31} \bowtie d_{21})) = 18 \), so the semijoin operation is used. Similarly, this is repeated for the schedules of attribute \( d_{i2} \) and the resulting solution is given below:
Proposed Methodology

Total time = \( C(18) + C(18) + C(0.11 \times 1700) \)
= \( 18 + 18 + 187 \)
= 223

Total time = \( C(19) + C(0.11 \times 1700) \)
= \( 18 + 187 \)
= 205

Total time = \( C(18) + C(18) + C(0.15 \times 0.11 \times 147) \)
\[
+ \quad C(0.11 \times 0.13 \times 1700)
\]
= \( 18 + 18 + 3 + 25 \)
= 64

Total time = \( C(18) + C(10.15 \times 125) + C(0.11 \times 0.13 \times 1700) \)
= \( 18 + 17 + 25 \)
= 60
The cost of each schedule for attribute $d_{11}$ is calculated and the best schedule is chosen. The best schedule $BEST_{11}$ is schedule $d_{31}'$. We repeat the process for attribute $d_{12}$, the best schedule $BEST_{12}$ is schedule $d_{22}'$. The $BEST_{ij}$ schedules are ordered on the total cost and integrated.
Total time = $C(18) + C(0.15 \times 125) + C(14) + C(0.11 \times 0.13 \times 0.03 \times 1700)$
= $18 + 17 + 14 + 1$
= 50

Choose the smallest of the integrated schedule as the solution of Algorithm AHY-HS for relation $R_1$.

The schedules for relations $R_2$ and $R_3$ are constructed in a similar way. The complete Algorithm AHY-HS strategy derived for the given query is
Based on the statistical query table given, when Algorithm AHY-HS was applied, it produced execution schedules with total cost of 191.

3.4 Comparison of Results

From the results generated in the illustrative example and by comparison, all
<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Feasible Solution (IFS)</td>
<td>7060</td>
</tr>
<tr>
<td>AHY Algorithm GENERAL (total time)</td>
<td>555</td>
</tr>
<tr>
<td>Algorithm R1</td>
<td>252</td>
</tr>
<tr>
<td>Algorithm R2</td>
<td>237</td>
</tr>
<tr>
<td>Algorithm AHY-H</td>
<td>192</td>
</tr>
<tr>
<td>Algorithm AHY-HS</td>
<td>191</td>
</tr>
</tbody>
</table>

The proposed algorithms (R1, R2, AHY-H & AHY-HS) outperform Algorithm GENERAL (total time) in terms of the cost of transmitting data during query processing.
Chapter 4
Evaluation

Because of the difficulty in obtaining a real database with a large set of queries to study the significance of the results, most evaluations of optimization algorithms have been done by theoretical analysis of their properties and expected performance. Due to this, a systematical approach\textsuperscript{11} which generates synthetic data and queries will be adopted in this thesis. The approach allows us to vary relation sizes, the selectivity, the numbers of join attributes, to have different data distributions and so on.

In this chapter, the performance of the proposed algorithms is evaluated against the AHY Algorithm. The AHY Algorithm is chosen as a benchmark for the following reasons:

\begin{itemize}
  \item In contrast to other recent algorithms, it is well documented. Thereby, a correct implementation and fair comparison between algorithms is possible.
  \item It is evident from the numerous citations in articles, that the AHY Algorithms are considered to be milestones in the field of distributed query optimization and are the best heuristics proposed to handle general queries.
\end{itemize}

\textsuperscript{11} Simulates a relational database environmental in form of a statistical table
4.1 Test Queries and Database

The objective of this thesis is to investigate the performance of the proposed algorithms with a wide range of select-project-join (SPJ) queries. The SPJ type of queries are considered since most relational queries can be expressed in that format [CP84]. A statistical table which consists of information about relations and attributes that are participating in the processing of queries is adopted. Such a representation facilitates the construction of a wide variety of test queries because constructing queries explicitly is unrealistic [Bod85].

The query parameters are as follows:

- Each query has between 3 and 6 relations and the number of joining attributes varies between 2 and 4. This gives us a total of 12 different types of test queries (e.g. 3 relations with 2 joining attributes, 3 relations with 3 joining attributes, etc.).

- The number of possible values in each domain varies between 500 and 1500 (all domains are integer based).

- The number of join attributes in each relation is set using a connectivity\(^{12}\) of 75%.

- Each relation has between 600 and 2000 tuples.

\(^{12}\) The number of joining attributes in each relation
Each relation has at least one non-joining attribute which is required at the query site.

The active domain values are distributed randomly.

A main advantage of adopting the statistical representation of queries is that only the relations that are needed to participate in the execution of a query are required to be constructed. The statistical table is constructed for the experimental setup given in Table 4.1, by giving the desired number of relations and the maximum number of join-attributes using a TCL/TK program called "qtool5.2" which is a modified version of the Wisconsin benchmark database[BDT83]. The program was developed by the database research team of the University of Windsor. The statistical information and the relations described in the statistical information are used as input files by a C program called "relbuilder.c" developed by Todd Bealor [Bea95] for fabricating relations in the statistical table with the chosen characteristics such as the number of participating relations, their respective sizes, the number of join attributes and their respective size and selectivities.

4.2 Experimental Results & Discussions

The results of evaluating the proposed algorithms are discussed in this section. For the purpose of our evaluation, the C programming language is used to program each of the proposed algorithms and the AHY Algorithm (total time). Three sets of
experiments will be performed varying the selectivity. Each experiment uses the parameters given below in Table 4.1 and is varied using the different selectivities given below in Table 4.2

<table>
<thead>
<tr>
<th>Experimental Setup</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of relations</td>
<td>3 - 6</td>
</tr>
<tr>
<td>Number of attributes</td>
<td>2 - 4</td>
</tr>
<tr>
<td>Attribute domain range</td>
<td>500 - 1500</td>
</tr>
<tr>
<td>Relation cardinality</td>
<td>600 - 2000</td>
</tr>
<tr>
<td>Query connectivity</td>
<td>75%</td>
</tr>
<tr>
<td>Distribution type</td>
<td>Random</td>
</tr>
</tbody>
</table>

Table 4.1 Experimental Setup

<table>
<thead>
<tr>
<th>Experiment#</th>
<th>Attribute selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.02 - 0.2</td>
</tr>
<tr>
<td>2</td>
<td>0.4 - 0.7</td>
</tr>
<tr>
<td>3</td>
<td>0.7 - 1.0</td>
</tr>
</tbody>
</table>

Table 4.2 Descriptions of each of the experiment with the selectivity range

The selectivity range between 0.02 to 0.2 is chosen because that is the only range where it is feasible to find some semijoin operation more cost-effective than hash-semijoin operations.
Evaluation

For each experiment, queries and corresponding relations are constructed. The queries are optimized and executed using Algorithm GENERAL (total cost) and proposed algorithms and their respective costs were recorded (see Appendix A).

A summary of the results from the three experiments are presented in Tables 4.3 – 4.5. For the purpose of our comparison, each entry in the table shows the average percentage improvement that the proposed algorithms and Algorithm GENERAL (total cost) achieves over the initial feasible solution (IFS). For example, type 3–2, means that we have 10 of the designated query type where each had 3 relations and 2 joining attributes, the connectivity is 75%, the domain values are distributed randomly; we have a selectivity range for the experiment. A 15 entry in the column means that the algorithm of that row has a 15% improvement over the IFS.
4.2.1 Results from Experiment #1

<table>
<thead>
<tr>
<th>Type</th>
<th>AHY</th>
<th>R1</th>
<th>R2</th>
<th>AHY-H</th>
<th>AHY-HS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 - 2</td>
<td>91.85</td>
<td>96.52</td>
<td>97.33</td>
<td>96.94</td>
<td>97.46</td>
</tr>
<tr>
<td>3 - 3</td>
<td>93.16</td>
<td>96.55</td>
<td>97.36</td>
<td>97.02</td>
<td>97.68</td>
</tr>
<tr>
<td>3 - 4</td>
<td>93.19</td>
<td>96.61</td>
<td>97.28</td>
<td>97.28</td>
<td>97.84</td>
</tr>
<tr>
<td>4 - 2</td>
<td>92.09</td>
<td>95.91</td>
<td>97.17</td>
<td>96.70</td>
<td>97.39</td>
</tr>
<tr>
<td>4 - 3</td>
<td>90.34</td>
<td>95.68</td>
<td>97.08</td>
<td>96.85</td>
<td>97.67</td>
</tr>
<tr>
<td>4 - 4</td>
<td>94.71</td>
<td>96.87</td>
<td>98.27</td>
<td>97.47</td>
<td>98.18</td>
</tr>
<tr>
<td>5 - 2</td>
<td>93.03</td>
<td>96.08</td>
<td>98.13</td>
<td>97.21</td>
<td>98.21</td>
</tr>
<tr>
<td>5 - 3</td>
<td>94.49</td>
<td>95.57</td>
<td>98.14</td>
<td>97.06</td>
<td>98.22</td>
</tr>
<tr>
<td>5 - 4</td>
<td>94.77</td>
<td>95.74</td>
<td>98.02</td>
<td>97.28</td>
<td>98.25</td>
</tr>
<tr>
<td>6 - 2</td>
<td>94.50</td>
<td>95.56</td>
<td>98.35</td>
<td>97.24</td>
<td>98.41</td>
</tr>
<tr>
<td>6 - 3</td>
<td>94.44</td>
<td>96.02</td>
<td>98.61</td>
<td>97.62</td>
<td>98.72</td>
</tr>
<tr>
<td>6 - 4</td>
<td>95.03</td>
<td>95.28</td>
<td>98.35</td>
<td>97.11</td>
<td>98.45</td>
</tr>
<tr>
<td>Average</td>
<td>93.47</td>
<td>96.03</td>
<td>97.84</td>
<td>97.15</td>
<td>98.04</td>
</tr>
</tbody>
</table>

Table 4.3 Average percentage improvement over the IFS using 0.02 – 0.2 selectivity

Based on an analysis of experiment #1, the following conclusions are drawn:

- Based on the results, it is only in this case did using both semijoin and hash-semijoin operators perform better than using only the hash-semijoin operator in the proposed replacement and heuristic algorithms.
- The experimental results demonstrate that AHY Algorithm (total cost) and the proposed algorithms have over 90% improvement over the IFS when the selectivity is between 0.02 and 0.2. The reason behind this is that there are few
tuples that are participating in the result of a query, so reducing the relations being sent to the query site using a semijoin or hash-semijoin operator is much better than sending the whole relations.

- When the selectivity is very low, there is no significant difference in improvement between the proposed algorithms and the AHY Algorithm. This suggests that the hypothesis of Tseng and Chen is not valid in this case, since replacing all or some of the semijoins with hash-semijoins did not lead to a significant improvement.
4.2.2 Results from Experiment #2

<table>
<thead>
<tr>
<th>Type</th>
<th>AHY</th>
<th>R1</th>
<th>R2</th>
<th>AHY-H</th>
<th>AHY-HS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 - 2</td>
<td>15.67</td>
<td>57.67</td>
<td>57.67</td>
<td>77.21</td>
<td>77.21</td>
</tr>
<tr>
<td>3 - 3</td>
<td>18.06</td>
<td>58.44</td>
<td>58.44</td>
<td>82.77</td>
<td>82.77</td>
</tr>
<tr>
<td>3 - 4</td>
<td>29.29</td>
<td>63.39</td>
<td>63.39</td>
<td>85.81</td>
<td>85.81</td>
</tr>
<tr>
<td>4 - 2</td>
<td>12.36</td>
<td>62.11</td>
<td>62.11</td>
<td>79.54</td>
<td>79.54</td>
</tr>
<tr>
<td>4 - 3</td>
<td>26.62</td>
<td>70.51</td>
<td>70.51</td>
<td>86.47</td>
<td>86.47</td>
</tr>
<tr>
<td>4 - 4</td>
<td>27.17</td>
<td>72.33</td>
<td>72.33</td>
<td>89.93</td>
<td>89.93</td>
</tr>
<tr>
<td>5 - 2</td>
<td>28.09</td>
<td>80.27</td>
<td>80.27</td>
<td>89.36</td>
<td>89.36</td>
</tr>
<tr>
<td>5 - 3</td>
<td>35.75</td>
<td>76.59</td>
<td>76.59</td>
<td>90.87</td>
<td>90.87</td>
</tr>
<tr>
<td>5 - 4</td>
<td>34.78</td>
<td>78.44</td>
<td>78.44</td>
<td>91.02</td>
<td>91.02</td>
</tr>
<tr>
<td>6 - 2</td>
<td>23.38</td>
<td>73.99</td>
<td>73.99</td>
<td>86.74</td>
<td>86.74</td>
</tr>
<tr>
<td>6 - 3</td>
<td>32.28</td>
<td>81.59</td>
<td>81.59</td>
<td>90.00</td>
<td>90.00</td>
</tr>
<tr>
<td>6 - 4</td>
<td>33.55</td>
<td>79.21</td>
<td>79.21</td>
<td>90.01</td>
<td>90.01</td>
</tr>
<tr>
<td>Average</td>
<td>26.42</td>
<td>71.21</td>
<td>71.21</td>
<td>86.65</td>
<td>86.65</td>
</tr>
</tbody>
</table>

Table 4.4 Average percentage improvement over the IFS using 0.4 – 0.7 selectivity

In this experiment, the performance of using an attribute selectivity between 0.4 and 0.7 is evaluated. Based on the results obtained from experiment #2, the following conclusions are drawn:

- The experimental results demonstrate that the two proposed heuristic algorithms (AHY-H and AHY-HS) give approximately a 60% improvement over the AHY Algorithm; and the proposed replacement algorithms, Algorithm R1
and R2 give approximately a 40% improvement, using a selectivity between 0.4 and 0.7 in both cases.

- None of the semijoin operations is found to be cost-effective in Algorithm R2. Hence, Algorithm R2 performs exactly as Algorithm R1. Thus, there is no difference in the percentage improvement obtained in both cases. This suggests that the hypothesis of Tseng and Chen is not valid in these circumstances, since we have found that it is always better to replace all the semijoins with hash-semijoins.

- Also, Algorithm AHY-H and Algorithm AHY-HS both have the same percentage improvement which also means that performing a semijoin operation is not found to be cost-effective when constructing the schedules in Algorithm AHY-HS.
### 4.2.3 Results from Experiment #3

<table>
<thead>
<tr>
<th>Type</th>
<th>AHY</th>
<th>R1</th>
<th>R2</th>
<th>AHY-H</th>
<th>AHY-HS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 - 2</td>
<td>-29.12</td>
<td>17.97</td>
<td>17.97</td>
<td>37.40</td>
<td>37.40</td>
</tr>
<tr>
<td>3 - 3</td>
<td>-27.50</td>
<td>18.76</td>
<td>18.76</td>
<td>38.81</td>
<td>38.81</td>
</tr>
<tr>
<td>3 - 4</td>
<td>-20.36</td>
<td>21.26</td>
<td>21.26</td>
<td>43.84</td>
<td>43.84</td>
</tr>
<tr>
<td>4 - 2</td>
<td>-25.44</td>
<td>20.64</td>
<td>20.64</td>
<td>41.03</td>
<td>41.03</td>
</tr>
<tr>
<td>4 - 3</td>
<td>-30.06</td>
<td>18.63</td>
<td>18.63</td>
<td>45.20</td>
<td>45.20</td>
</tr>
<tr>
<td>4 - 4</td>
<td>-23.25</td>
<td>20.81</td>
<td>20.81</td>
<td>61.66</td>
<td>61.66</td>
</tr>
<tr>
<td>5 - 2</td>
<td>-34.20</td>
<td>15.90</td>
<td>15.90</td>
<td>52.47</td>
<td>52.47</td>
</tr>
<tr>
<td>5 - 3</td>
<td>-29.98</td>
<td>18.45</td>
<td>18.45</td>
<td>63.36</td>
<td>63.36</td>
</tr>
<tr>
<td>5 - 4</td>
<td>-21.96</td>
<td>21.17</td>
<td>21.17</td>
<td>69.02</td>
<td>69.02</td>
</tr>
<tr>
<td>6 - 2</td>
<td>-36.62</td>
<td>16.63</td>
<td>16.63</td>
<td>52.10</td>
<td>52.10</td>
</tr>
<tr>
<td>6 - 3</td>
<td>-27.53</td>
<td>21.68</td>
<td>21.68</td>
<td>67.49</td>
<td>67.49</td>
</tr>
<tr>
<td>6 - 4</td>
<td>-21.85</td>
<td>22.10</td>
<td>22.10</td>
<td>66.59</td>
<td>66.59</td>
</tr>
<tr>
<td>Average</td>
<td>-27.32</td>
<td>19.50</td>
<td>19.50</td>
<td>53.25</td>
<td>53.25</td>
</tr>
</tbody>
</table>

*Table 4.5 Average percentage improvement over the IFS using 0.7 - 1.0 selectivity*

In this experiment, the performance of using an attribute selectivity between 0.7 and 1.0 is evaluated. Based on the results obtained from experiment #3, the following conclusions are drawn:

- When the selectivity is very high, the AHY Algorithm (total time) performs worse than the IFS by approximately 27%. The reason behind this result is that over 70% of the tuples in each relation are required at the query site so
using a semijoin operator as a reducer simply increases the amount of data transferred.

- Algorithms R1 and R2 perform better than the AHY Algorithm by over 45%.
- None of the semijoin operations is found to be cost-effective in Algorithm R2. Hence, Algorithm R2 performs exactly the same as Algorithm R1. Thus, there is no difference in the percentage improvement obtained in both cases. This suggests again that the hypothesis of Tseng and Chen is not valid in these circumstances, since we have found that it is always better to replace all the semijoins with hash-semijoins.

- Algorithms AHY-H and AHY-HS also perform better than the AHY Algorithm, by over 80%.
- Also, Algorithm AHY-H and Algorithm AHY-HS both have the same percentage improvement which also means that performing a semijoin operation is not found to be cost-effective when constructing the schedules in Algorithm AHY-HS.

- The heuristic algorithms proposed, Algorithm AHY-H and Algorithm AHY-HS, both outperform all the other algorithms in terms of the cost of data transmitted during query processing.

### 4.3 Research Summary

- In experiment #1, it is demonstrated that only in the selectivity range 0.02 to
0.2 does it ever become cost-effective to retain some of the semijoin operations as part of the schedules or in deriving a schedule.

☐ Except for the queries where the selectivity is in the range 0.02 to 0.2, it is always better to replace all semijoins with hash-semijoins. Since the likelihood of such queries is very small and the difference in performance is not significant, then the hypothesis can be replaced with “replacing all semijoins with hash-semijoins may lead to significant improvement in the performance of a semijoin-based algorithm”.

☐ The attribute selectivity between 0.2 and 0.4 is not considered significant because sample runs showed that none of the semijoin operations would be found cost-effective in Algorithm R1 and AHY-HS, which is the same case as using selectivity range 0.4 to 1.0.

☐ In experiments #2 and #3, we show that it is always better to replace all semijoins with hash-semijoins. This does lead to a significant improvement in the performance of a semijoin-based algorithm.

☐ Also experiments #2 and #3, Algorithm AHY-H and AHY-HS, both have the same percentage improvement over the AHY Algorithm (total time) when the selectivity range is between 0.4 to 1.0. This implies that Algorithm AHY-HS reverts to Algorithm AHY-H so that only hash-semijoin operators are used in deriving schedules.

☐ Overall, Algorithm AHY-H and Algorithm AHY-HS both have the best
performance in terms of the total cost of transmitting data during query processing. On average, they provide a reduction of between 50% to 98% (approximately) over the initial feasible solution.

- From the results generated, using only hash-semijoin operators in deriving schedules is suggested to be more cost-effective in our experimental framework because only in the case where the selectivity is very low did using both semijoin and hash-semijoin operators perform better.

- Also, using only hash-semijoin operators in deriving schedules is suggested to be more cost-effective in our experimental framework than using only the traditional semijoin operator or the hypothesis suggested by Tseng and Chen.
5.1 Overview

The principal problem in evaluating a relational query on a distributed database system is that relations must be joined and these relations may reside at different sites. For the join to be performed, one relation must be shipped to the other site. For a given query, there exist several strategies that a distributed DBMSs may choose to execute in order to generate results. All these strategies produce the same result but incur different query execution costs. Performance of a relational distributed database system in executing a query is critically dependent upon the ability of the query processor to derive efficient execution strategies. These processor differ in terms of the distributed database environments, assumptions made and techniques used for processing queries. Finding an optimal execution strategy is NP-hard [WC96], therefore most existing algorithms are just heuristic. The efficiency of the heuristic depends heavily on the reducer used.

5.2 Conclusions

In this thesis, we have presented and evaluated two replacement algorithms and two heuristic algorithms for minimizing the total cost of transmitting data over the network during distributed query processing. Algorithm R1, which is
a replacement algorithm, replaces all the semijoin operation in a schedule with hash-semijoin operation. Algorithm R2, which is also a replacement algorithm but based on the hypothesis suggested by Tseng and Chen, replaces some of the semijoin operation in a schedule for more cost-effectiveness with hash-semijoin operation. Algorithm AHY-H, which is a static heuristic algorithm which uses exactly the same heuristic as Algorithm GENERAL (total time) but instead of deriving schedules using semijoin operation it uses hash-semijoin operation. Algorithm AHY-HS which is also a static heuristic algorithm similar to both Algorithm GENERAL (total time) and Algorithm AHY-H in terms of the heuristic used but an improvement. In Algorithm AHY-HS, both the semijoin operation and the hash-semijoin operation are adopted in a cost-effective way to determine distribution strategy for a general query.

The performance of each of the proposed algorithms is tested extensively against the AHY Algorithm (total time). Results from our evaluation show that both Algorithm R1 and R2, perform better than the AHY Algorithm and there is no significant difference between them using our experimental framework. Both the heuristic algorithms perform better than the AHY Algorithm and the proposed replacement algorithms. Algorithm AHY-HS reverts to Algorithm AHY-H so they have exactly the same performance but only in the case where the selectivity is very low did Algorithm AHY-HS perform better than Algorithm AHY-H, even though the difference is not significant. Based on our findings, we suggest using
only hash-semijoin operators in deriving schedules (Algorithm AHY-H) since it is more cost-effective than the hypothesis suggested in [TC92] by Tseng and Chen and the AHY Algorithm (total time).

5.3 Recommendations for Future Research

Based on the assumptions stated initially and observations made during the investigation, the following recommendations are proposed for further study:

[i] In this thesis, a way of reducing the amount of data transmitted between sites is addressed but may be at the expense of additional local processing. Therefore, incorporating the local processing cost in the cost function of the proposed algorithm is critical for the correct estimation of the execution costs when determining the optimal execution strategy.

[ii] Single Bloom filters are used for transmission in this thesis but may later prove ineffective in cases where the size of the Bloom filter is very large. Multiple Bloom filters are suggested for further experimentation in the proposed algorithms.
Bibliography


Appendix A

Result Summaries

A.1 Total Cost

In this appendix, the results of the total cost analysis have been summarized into the following tables. The query type (number of relations and joining attributes) are given in column 1. The average total cost of executing 10 of the query type with the different algorithms are presented in the other columns. The overall average total cost over all of the query types are given at the last row of the table.
<table>
<thead>
<tr>
<th>Type</th>
<th>IFS</th>
<th>AHY</th>
<th>R1</th>
<th>R2</th>
<th>AHY-H</th>
<th>AHY-HS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 - 2</td>
<td>4051</td>
<td>330</td>
<td>141</td>
<td>108</td>
<td>124</td>
<td>103</td>
</tr>
<tr>
<td>3 - 3</td>
<td>3829</td>
<td>262</td>
<td>132</td>
<td>101</td>
<td>114</td>
<td>89</td>
</tr>
<tr>
<td>3 - 4</td>
<td>4158</td>
<td>283</td>
<td>141</td>
<td>113</td>
<td>113</td>
<td>90</td>
</tr>
<tr>
<td>4 - 2</td>
<td>5334</td>
<td>422</td>
<td>218</td>
<td>151</td>
<td>176</td>
<td>139</td>
</tr>
<tr>
<td>4 - 3</td>
<td>5143</td>
<td>497</td>
<td>222</td>
<td>150</td>
<td>162</td>
<td>120</td>
</tr>
<tr>
<td>4 - 4</td>
<td>5331</td>
<td>282</td>
<td>167</td>
<td>92</td>
<td>135</td>
<td>97</td>
</tr>
<tr>
<td>5 - 2</td>
<td>6095</td>
<td>425</td>
<td>239</td>
<td>114</td>
<td>170</td>
<td>109</td>
</tr>
<tr>
<td>5 - 3</td>
<td>6024</td>
<td>332</td>
<td>267</td>
<td>112</td>
<td>177</td>
<td>107</td>
</tr>
<tr>
<td>5 - 4</td>
<td>6577</td>
<td>344</td>
<td>280</td>
<td>130</td>
<td>179</td>
<td>115</td>
</tr>
<tr>
<td>6 - 2</td>
<td>7563</td>
<td>416</td>
<td>336</td>
<td>125</td>
<td>209</td>
<td>120</td>
</tr>
<tr>
<td>6 - 3</td>
<td>7482</td>
<td>416</td>
<td>298</td>
<td>104</td>
<td>178</td>
<td>96</td>
</tr>
<tr>
<td>6 - 4</td>
<td>7290</td>
<td>362</td>
<td>344</td>
<td>120</td>
<td>211</td>
<td>113</td>
</tr>
<tr>
<td>Average</td>
<td>5739.75</td>
<td>364.25</td>
<td>232.08</td>
<td>118.33</td>
<td>162.33</td>
<td>108.17</td>
</tr>
</tbody>
</table>

Table A.1 Average total cost using 0.02 – 0.2 selectivity
<table>
<thead>
<tr>
<th>Type</th>
<th>IFS</th>
<th>AHY</th>
<th>R1</th>
<th>R2</th>
<th>AHY-H</th>
<th>AHY-HS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 - 2</td>
<td>3976</td>
<td>3353</td>
<td>1683</td>
<td>1683</td>
<td>906</td>
<td>906</td>
</tr>
<tr>
<td>3 - 3</td>
<td>4069</td>
<td>3334</td>
<td>1691</td>
<td>1691</td>
<td>701</td>
<td>701</td>
</tr>
<tr>
<td>3 - 4</td>
<td>4138</td>
<td>2926</td>
<td>1515</td>
<td>1515</td>
<td>587</td>
<td>587</td>
</tr>
<tr>
<td>4 - 2</td>
<td>5395</td>
<td>4728</td>
<td>2044</td>
<td>2044</td>
<td>1104</td>
<td>1104</td>
</tr>
<tr>
<td>4 - 3</td>
<td>5579</td>
<td>4094</td>
<td>1645</td>
<td>1645</td>
<td>755</td>
<td>755</td>
</tr>
<tr>
<td>4 - 4</td>
<td>5789</td>
<td>4216</td>
<td>6102</td>
<td>1602</td>
<td>583</td>
<td>583</td>
</tr>
<tr>
<td>5 - 2</td>
<td>6529</td>
<td>4695</td>
<td>1288</td>
<td>1288</td>
<td>695</td>
<td>695</td>
</tr>
<tr>
<td>5 - 3</td>
<td>6889</td>
<td>4426</td>
<td>1613</td>
<td>1613</td>
<td>629</td>
<td>629</td>
</tr>
<tr>
<td>5 - 4</td>
<td>6763</td>
<td>4411</td>
<td>1458</td>
<td>1458</td>
<td>607</td>
<td>607</td>
</tr>
<tr>
<td>6 - 2</td>
<td>8042</td>
<td>6162</td>
<td>2092</td>
<td>2092</td>
<td>1066</td>
<td>1066</td>
</tr>
<tr>
<td>6 - 3</td>
<td>8063</td>
<td>5460</td>
<td>1484</td>
<td>1484</td>
<td>806</td>
<td>806</td>
</tr>
<tr>
<td>6 - 4</td>
<td>8062</td>
<td>5357</td>
<td>1676</td>
<td>1676</td>
<td>805</td>
<td>805</td>
</tr>
<tr>
<td>Average</td>
<td>6107.83</td>
<td>4430.17</td>
<td>1649.25</td>
<td>1649.25</td>
<td>770.33</td>
<td>770.33</td>
</tr>
</tbody>
</table>

Table A.2 Average total cost using 0.4 – 0.7 selectivity
<table>
<thead>
<tr>
<th>Type</th>
<th>IFS</th>
<th>AHY</th>
<th>R1</th>
<th>R2</th>
<th>AHY-H</th>
<th>AHY-HS</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 - 2</td>
<td>4457</td>
<td>5755</td>
<td>3656</td>
<td>3656</td>
<td>2790</td>
<td>2790</td>
</tr>
<tr>
<td>3 - 3</td>
<td>4473</td>
<td>5703</td>
<td>3634</td>
<td>3634</td>
<td>2737</td>
<td>2737</td>
</tr>
<tr>
<td>3 - 4</td>
<td>4642</td>
<td>5587</td>
<td>3655</td>
<td>3655</td>
<td>2607</td>
<td>2607</td>
</tr>
<tr>
<td>4 - 2</td>
<td>5669</td>
<td>7111</td>
<td>4499</td>
<td>4499</td>
<td>3343</td>
<td>3343</td>
</tr>
<tr>
<td>4 - 3</td>
<td>5992</td>
<td>7754</td>
<td>4851</td>
<td>4851</td>
<td>3267</td>
<td>3267</td>
</tr>
<tr>
<td>4 - 4</td>
<td>6421</td>
<td>7914</td>
<td>5085</td>
<td>5085</td>
<td>2462</td>
<td>2462</td>
</tr>
<tr>
<td>5 - 2</td>
<td>7112</td>
<td>9544</td>
<td>5981</td>
<td>5981</td>
<td>3380</td>
<td>3380</td>
</tr>
<tr>
<td>5 - 3</td>
<td>7399</td>
<td>9617</td>
<td>6034</td>
<td>6034</td>
<td>2711</td>
<td>2711</td>
</tr>
<tr>
<td>5 - 4</td>
<td>7799</td>
<td>9512</td>
<td>6148</td>
<td>6148</td>
<td>2416</td>
<td>2416</td>
</tr>
<tr>
<td>6 - 2</td>
<td>8933</td>
<td>12204</td>
<td>7447</td>
<td>7447</td>
<td>4279</td>
<td>4279</td>
</tr>
<tr>
<td>6 - 3</td>
<td>9392</td>
<td>11978</td>
<td>7356</td>
<td>7356</td>
<td>3053</td>
<td>3053</td>
</tr>
<tr>
<td>6 - 4</td>
<td>9208</td>
<td>11220</td>
<td>7173</td>
<td>7173</td>
<td>3076</td>
<td>3076</td>
</tr>
<tr>
<td>Average</td>
<td>6788.92</td>
<td>8658.25</td>
<td>5459.92</td>
<td>5459.92</td>
<td>3010.08</td>
<td>3010.08</td>
</tr>
</tbody>
</table>

Table A.3 Average total cost using 0.7 – 1.0 selectivity
Vita Auctoris

Olumuyiwa Ogunbadejo was born in 1972 in London, Ontario. He graduated from Federal Government College Lagos in 1989. From there he went on to the University of Lagos, Nigeria where he obtained a B. Sc. in Computer Science in 1995. He is currently a candidate for the Master’s degree in Computer Science at the University of Windsor and hopes to graduate in the Spring of 1998.