The use of multiple Bloom-filters to minimize data transfer during distributed query processing.

Jianwei. Wang
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/4334
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.

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

UMI
THE USE OF MULTIPLE BLOOM-FILTERS TO MINIMIZE DATA TRANSFER DURING DISTRIBUTED QUERY PROCESSING

By
Jianwei Wang

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
2000
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.

0-612-62298-3
Abstract

Query processing in distributed database system requires the transmission of data between computers in network. Therefore, query optimization in distributed database system is an important research issue. Since employing the optimization for general query is NP-Hard, heuristics are applied to find a cost-effective and efficient processing strategy. The challenge is how to efficiently minimize either transmission time or local processing cost in a query process.

In the thesis, we propose a new reduction approach to significantly minimize data transmission time. The algorithm [32] is used to process general queries by simply substituting single bloom filter that is based on perfect hashing in the reduction approach [32] with multiple Bloom filters which are based on non-perfect hash functions. Our approach aims to minimize data transmission time. The evaluation of our application is against the reducer [32]. An analysis of how the number of bloom-filters affects the performance of the algorithm [32] is provided in the thesis. An amount of experimental results will be used to evaluate the performance of our reduction approach. Compared to the approach in paper [32], our reduction approach provides a more practical, cost effective and efficient processing query solution.
ACKNOWLEDGEMENTS

I am indebted to a number of people who provided helpful and constructive criticism of the thesis. A big thank-you to my supervisor Dr. Joan Morrissey for her comments, advice, encouragement and patience. Thanks also to Dr. Christie Ezeife, Dr. Quan Wen and Dr. Yung H. Tsin who reviewed my thesis and provided valuable comments. Thanks also to Dr. Kaizhong Zhang who provided constructive advice. I also thank to Dr. Wendy K. Osborn for her introduction of experiments.

Finally, thanks to my sister, Dr. Jinfei Wang who provided encouragement and inspiration. And thanks to everybody who helps me.
# TABLE OF CONTENTS

ABSTRACT ................................................................. iv

ACKNOWLEDGEMENTS ......................................................... v

LIST OF TABLES ........................................................... x

LIST OF FIGURES ........................................................... xi

Chapter 1  INTRODUCTION ....................................................... 1

1.1 Review of database systems .............................................. 1

1.2 Distributed database systems .......................................... 2

  1.2.1 What is Distributed database systems? .......................... 2

  1.2.2 History of distributed database system ......................... 4

  1.2.3 Advantages of DDBMS ............................................. 5

  1.2.4 Disadvantages of DDBMS .......................................... 6

1.3 Query optimization overview .......................................... 6

  1.3.1 Overview ................................................................ 6

  1.3.2 Query optimization algorithm .................................... 7

  1.3.3 Cost model ........................................................... 7

Chapter 2  RELATED WORK ..................................................... 10

2.1 Optimization timing ..................................................... 10

  2.1.1 Static strategy ........................................................ 10

  2.1.2 Dynamic strategy .................................................... 10

  2.1.3 Hybrid strategy ....................................................... 11
2.2 JOIN operation and application of JOIN ........................................ 11
  2.2.1 JOIN operation ............................................................... 11
  2.2.2 Application of JOIN in Distributed DBMS ......................... 11
    2.2.2.1 R* algorithm .......................................................... 11
    2.2.2.2 Dynamic Join strategies with a tree model .................. 13
  2.3 Semijoin operation and application of Semijoin ...................... 14
    2.3.1 Semijoin operation .................................................... 14
    2.3.2 Application of Semijoin in Distributed DBMS .................. 16
      2.3.2.1 SDD-1 reducer ...................................................... 16
      2.3.2.2 The Algorithm AHY(GENERAL) .................................. 16
      2.3.2.3 Two-Way Semijoins .............................................. 17
      2.3.2.4 Domain-Specific Semijoin ..................................... 19
  2.4 Combination of Join and Semijoin and its application .............. 19
    2.4.1 Combination of join and semi-join operations .................. 19
  2.5 Bloom filter based Semijoin in Distributed DBMS .................... 21
    2.5.1 Operation .............................................................. 22
    2.5.2 Hash-Semijoins ....................................................... 23
    2.5.3 Algorithm X ............................................................ 23
    2.5.4 A filter based algorithm ........................................... 24

CHAPTER 3 ASSUMPTIONS .......................................................... 26

CHAPTER 4 INTRODUCTION OF OUT REDUCTION APPROACH ..................... 27
  4.1 The difference between the two reduction approaches .............. 27
4.2 How does our reducer work? ........................................ 28
4.3 Some related concepts ............................................. 29
  4.3.1 Query graph .................................................... 30
  4.3.2 Adjacency Matrix .............................................. 30
  4.3.3 Adjacency List .................................................. 31
  4.3.4 Inverted List ................................................... 32
  4.3.5 Queue .......................................................... 32
  4.3.6 Filter Rule ..................................................... 32
  4.3.7 Adjacency Matrix ............................................. 30
4.4 Detailed description of our reducer .............................. 33
4.5 Constructing efficient hash functions ........................... 34
4.6 An example ....................................................... 35
4.7 Summary .......................................................... 47

CHAPTER 5 PERFORMANCE EVALUATION .......................... 48
5.1 Methodology ....................................................... 48
  5.1.1 The factors affecting our evaluation ......................... 48
  5.1.2 Evaluating experimental results ............................. 49
5.2 Experimental Plan ................................................ 50

CHAPTER 6 ANALYSIS OF EXPERIMENTAL RESULTS ............ 52
6.1 Experimental results of reducer [32] ........................... 52
6.2 Experimental results of our reducer ............................ 53
  6.2.1 Experimental results of our reducer by one filter ......... 54
  6.2.2 Experimental results of our reducer by two filters ....... 56
6.2.3 Experimental results of our reducer by three filters .......... 57
6.2.4 Experimental results of our reducer by four filters .......... 59
6.2.5 Experimental results of our reducer by five filters .......... 60
6.2.6 Experimental results of our reducer by six filters .......... 62

6.3 Summary ................................................................. 62
    6.3.1 Comparison according to number of bloom-filters and connectivity .. 62
    6.3.2 Comparison according to number of relations and selectivity .......... 63
    6.3.3 Comparison of different query types ................................ 64
    6.3.4 Comparison according to number of join attributes and number
                        of bloom-filters .............................................. 65
    6.3.5 Comparison according to number relations and number of bloom filters... 67

CHAPTER 7 CONCLUSION AND FUTURE RESEARCH ....................... 68
    7.1 Conclusion .......................................................... 68
    7.2 Future Work ........................................................ 71

APPENDIX A ............................................................... 72

SELECTED BIBLIOGRAPHY ............................................... 74

VITA AUCTORIS ............................................................. 83
# List of Tables

Table 1: Adjacency Matrix ........................................... 31
Table 2: The results after applying the reducer [32] .................. 53
Table 3: The results of applying one bloom-filter ................... 54
Table 4: The results of applying two bloom-filter ................... 56
Table 5: The results of applying three bloom-filter ................. 58
Table 6: The results of applying four bloom-filter ................ 60
Table 7: The results of applying five bloom-filter ................. 61
Table 8: The reduction result of varying connectivity and number of bloom-filers 63
Table 9: Reduction results of applying one to six filters with varying selectivity ................................. 64
Table 10: Reduction results varying querying types after applying one to six bloom filters ........................................ 65
Table 11: Reduction results of varying number of join attributes and number of bloom-filters ..................... 66
Table 12: Reduction results of varying number of relations and number of bloom-filters ............................... 67
Table 13: Final reduction results of applying one to six bloom-filters ....................... 70
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Distributed computer systems</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>Distributed database options</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>Example of total cost mode</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Example of response time cost model</td>
<td>8</td>
</tr>
<tr>
<td>5: Join</td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>A tree model for join strategies</td>
<td>14</td>
</tr>
<tr>
<td>7</td>
<td>Semijoin cycle</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>Two-Way semijoin</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>A bloom Filter with one hash function</td>
<td>22</td>
</tr>
<tr>
<td>10</td>
<td>Example of use of multi-filters</td>
<td>28</td>
</tr>
<tr>
<td>11</td>
<td>Example Relations</td>
<td>29</td>
</tr>
<tr>
<td>12</td>
<td>Query graph</td>
<td>30</td>
</tr>
<tr>
<td>13</td>
<td>Adjacency List</td>
<td>32</td>
</tr>
<tr>
<td>14</td>
<td>Query graph after deleting node R1 and edge A</td>
<td>36</td>
</tr>
<tr>
<td>15</td>
<td>Query graph after deleting node R2 and edge E</td>
<td>38</td>
</tr>
<tr>
<td>16</td>
<td>Query graph after deleting note R4 and edge D, F and G</td>
<td>40</td>
</tr>
<tr>
<td>17</td>
<td>Query graph after deleting note R4 and edge D, F and G</td>
<td>42</td>
</tr>
<tr>
<td>18</td>
<td>Reduced relations</td>
<td>46</td>
</tr>
<tr>
<td>19</td>
<td>Result of joining the reduced relations</td>
<td>46</td>
</tr>
</tbody>
</table>
CHAPTER 1 INTRODUCTION

A Distributed database system allows data to be distributed and managed on a network. There are many advantages of Distributed Database Management Systems over traditional Centralized Database System. However optimizing large queries in DDMS is difficult. In this paper, a new reduction approach is proposed to minimize transmission cost.

1.1 Review of database systems

The database technology for information systems has undergone four generations of evolution. Today most companies are using the fourth generation database systems. And the fifth as next generation is currently under development.

- The first generation was file systems, such as ISAM and VSAM.
- The second generation was hierarchical database systems, such as IMS and System 2000.
- The third generation was CODASYL database systems, such as IDS, TOTAL, ADABAS, IDMS, etc.
- The fourth generation was relational database, such as ORACLE, DB2 and SYbase.

The second and the third generation systems were able to share an integrated database among many users within an application environment. They lacked data independence and had tedious navigational access to the database. The fifth
generation which is Object-Oriented Database System supports data independence, but it is still in construction.

Today networks have played an important role in our real life. Moreover, the internet has emerged as a remarkable network information resource that is increasingly being used on a commercial business such as E-Commerce. Due to the distributed data on different sites, distributed database systems have emerged quickly, and have been widely been used in business. Within our research, we focus on distributed relational database systems.

1.2 DISTRIBUTED Database Systems

1.2.1 What is a DISTRIBUTED database system?

In the following diagram. The A, B, C, D, E are local sites. The lines represent network connections.

![Diagram of distributed computer systems]

Figure 1: Distributed computer systems

A Distributed Database System [22] is a single logical database system where relations are spread physically among multiple independent processing locations that are connected by a network.
• Distributed databases have been chosen by most computer systems so far because of their performance, reliability, availability, efficiency of processing and modularity. For example, local business wants control over data from different locations and to consolidate data across local databases for integrated decision making. Distributed databases can reduce telecommunications costs and reduce the risk of telecommunications failures.

![Distributed Database Diagram]

**Figure 2: Distributed database options**

• Distributed Database Options (Figure 2).

1) Homogeneous - Same DBMS at each node. Data is distributed across all the nodes. All data is managed by the distributed DBMS (no exclusively local data). All accesses are through one global schema. The global schema is the union of all local schemas.

• Autonomous - Independent DBMSs.
• Non-autonomous - Central, coordinating DBMS.

2) Heterogeneous - Different DBMSs at different nodes.

• Gateways - Simple paths are created to other databases without the benefits of one logical database.

• Systems - Supports some or all of the functionality of one logical database.
  (a) Full DBMS Functionality – include all distributed database functions.
  (b) Partial - Multi-database - Some distributed database functions.

A distributed database system [12, 37] can be defined as consisting of a collection of data with different parts under the control of separate DBMSs running on independent computer systems which could run on different platforms. All the computers are interconnected and each system has autonomous processing capability serving local applications. Each system participates, as well, in the execution of one or more global applications. Such applications require data from more than one site since data is geographically distributed, there are two kinds of geographic environments:

(1) Wide area network (WAN) - Point to point networks

Communication cost will dominate, ignore all other cost factors. Low bandwidth, low speed, high protocol overhead. In the research, we consider only WAN.
(2) Local area networks (LAN). All other cost factors except Communication cost could not be ignored.

1.2.2 History of Distributed Database System

The concepts behind distributed DBMS were pioneered during the late 1970’s in the IBM research project R*. IBM’s subsequent delivery of distributed DBMS products has been part of a 10 year evolving technology known as DRDA (distributed relational data architecture). DRDA at this time is largely an approach for integrating data sets across the different versions of DB2 that run on AIX, OS/2, OS/400, VM and MVS. DRDA has been published and IBM encourages other DBMS vendors to participate as client or server sites.

The first well-publicized distributed DBMS product was INGRES/Star, which was announced in 1987. Oracle also announced distributed DBMS capabilities in 1987. The first Oracle product to reasonably support distributed database processing is Oracle 7.x, which has been in the market since 1993.

1.2.3 Advantages of Distributed Database:

- Increased reliability and availability.
- Improved local control over data.
- Modular
- Lowered communication costs.
- Faster response for certain queries.
1.2.4 Disadvantages of the distributed Database

The disadvantages of the distributed approach to DBMS implementation are its cost and complexity. A distributed system, which hides its distributed nature from the end user, is more complex than the centralized system. Increased complexity means that the maintenance costs of the system are higher than those for centralized DBMS. The parallel nature of the system means that errors are harder to avoid and those in the applications are difficult to pinpoint. In addition, the distributed system, by its very nature, entails a large communication overhead in coordinating messages between the different sites. For general queries with arbitrary complexity to generate an optimal processing strategy in distributed database system is called NP-hard. That means we have to generate all possible query plan and chose the "best". It is inefficient and very expensive due to searching for the plan itself. Therefore, the computationally feasible algorithms to generate the processing strategies for general queries depend on heuristics. The objective of our research is to propose a general and practical heuristic processing strategy to optimize query processing and to reduce cost and complexity.

1.2.5 Requirements for a Distributed DBMS

- Ability to locate data with a distributed data dictionary.
- Determine the location from where to retrieve data and the location at which to process each part of a distributed query.
• Heterogeneous DBMS translation.
• Security, concurrency and failure recovery.
• Consistency of replicated data.
• Query optimization.

1.3 Query optimization overview

Query optimization [12, 37] is essential if a DBMS is to achieve acceptable performance and efficiency. Relational database systems are based on the relational model and relational algebra. The systems have the strength that their relational expressions are at a sufficiently high level, so that query optimization is feasible. For a given query, the system can decide how the query should be executed. In non-relational systems, user requests are low level and optimization is done manually by the user while the system can not help.

1.3.1 Query optimization algorithm

In a distributed relational database system [40], distributed query processing [5,8,19,24,30,43,45] requires shipping the relations between different sites. To reduce the data transmission cost and local costs, lots of query optimization algorithm have been proposed. Some of them will be introduced in chapter 2.

1.3.2 Cost model

Initially, cost is calculated by time. If we assume the transmission proportion is constant, which is denoted as C1. And the start up cost of initiating transmission is the same for all transactions, which is denoted as C0. However, since transmission domain the cost. We can simply ignore C0. Also assume an amount of data from site A is X(Figure 3). Then the cost of transferring data from site A to site B is (C1 * X). Assume Y is the amount of data shipping from site B to site C. Then the cost from site A to site C is (C1
* (X + Y)). Since C1 is constant, for any cost for any two sites data shipping. The cost can be calculated only depending on amount of transferred data. The cost for replying a query includes total cost model and response time model.

1) **Total cost model** [2, 26, 47]: In relational distributed databases a query cost consists of local cost and transmission cost.

- **local cost include:** [23]
  
  a) **Storage cost:** cost of writing data to secondary storage. I/O cost.
  
  b) **Secondary storage access cost:** The cost of reading data from secondary storage.
  
  c) **Computation cost:** CPU cost.

![Diagram](Site A X Site B Y Site C)

**Figure 3:** Example of total cost model

- **Transmission cost:** the cost of transmitting data from one site to another site.

Transmission cost determines major cost of query processing in distributed scenario. It is the cost of transmitting data from the site they are stored to the site where computations are performed and results are presented. Transmission cost is often assumed to be linear function of the number of data transmitted.

Today, more and more powerful computers are widely used. A typical ratio between the transmission cost and local cost for one page is about 20:1 [50]. It has been assumed that transmission costs were the predominant cost and
hence much of the research on query optimization has concentrated on near-optimal strategies, that is only transmission costs are considered.

![Diagram of sites A, B, and C with connections labeled X and Y.]

Figure 4: Example of response time cost model

2) Response time cost model: The time from starting a query to when the answer is produced. This involved in parallel data transfer. Suppose X is the size of data shipped from site A to site B. Suppose Y is the amount of data shipping from site B to site C (Figure 4). The response time cost is the maximum of \((C_0 + C_1 \times X)\) and \((C_0 + C_1 \times Y)\).

Total cost model is used in the research to simplify cost calculation. We only use total cost model for estimation in this paper. The rest of the paper is organized as follow: Chapter 2 introduce related work. Chapter 3 states the assumption for our experiments. In Chapter 4 we introduce our reduction approach. Thenext chaper explain how to evaluate our reducer performance. Chapter 6 analyze our experimental results. Finally, we present conclusions and future work.
CHAPTER 2 RELATED WORK

Some related work with our reduction approach will be introduced in the chapter.

2.1 Optimization timing: Static and dynamic

There are three kinds of processing optimization according to time, which are Static, Dynamic and hybrid.

2.1.1 Static strategy

The processing strategy [25] remains the same throughout its execution [8]. Static optimization select an access path for a query while the query is compiled [50]. It is completely determined before execution. The same access plan is repetitively used in run time until it is recompiled. Hence, the optimization overhead is zero at run time. The disadvantage is that it is hard to estimate the size of the intermediate results.

2.1.2 Dynamic strategy

This is run time optimization [7, 25, 27, 28]. If the database configuration and statistics could be changed dynamically, static optimization strategy may not be optimal any more. Therefore, the query execution plans should be re-optimized whenever needed. The algorithm’s disadvantage is optimization overhead, because the access path needs to be reconstructed [50] at execution time.

2.1.3 Hybrid strategy

At compile time, static algorithm is applied. But if relation sizes are greater than threshold, re-optimize the queries at run time.
2.2 JOIN operation and application of JOIN

2.2.1 JOIN operation

In relational algebra, the join [49] of relation A on attribute X with relation B on attribute Y yields the set of all tuples 't' such that 't' is a concatenation of a tuple 'a' belonging to A and tuple 'b' belonging to B and the predicate 'a.X comp b.Y' evaluates to true (attributes A.X and B.Y should be defined on the same domain). If the 'comp' operator is '=' then it is an equi-join and thus it must include two identical attributes. If one of these is removed (using a projection) then the result is a natural join - this is the most important kind of join. JOINs [14, 18, 22, 36, 39, 41] are represented in SQL by the SELECT statement. In the paper we only consider equi-join. Joins are written in shorthand form e.g. A join B, (or A ÷ B) and can be represented diagrammatically as Figure 5. In distributed systems, if relations A and B are not in the same location, we have to ship data in one location to another. The sequence of operations could be used to optimize queries. Most query processing strategy employs the query tree.

2.2.2 Application of JOIN in Distributed DBMS

2.2.2.1 JOIN is basis of SEMIJOIN. The algorithm is simple and is used by the optimizer of the first distributed DBMS – R*.

In 1979, R* [21, 29] is an experimental adaptation to a distributed environment of System R [21] and it is developed at IBM San Jose Research Laboratory. Total cost model is employed to estimate reduction performance. And R* algorithm
minimizes the cost of distributed join query. The cost include local processing, I/O, and communication cost. It support static strategy.

![Diagram of Relation R1(Site 1) and Relation R2(Site 2)]

Figure 5: Join

R* optimizer optimizes the entire query before execution of the query begins. It happens either immediately before execution starts or at compile time. The JOIN algorithm covered five aspects [21]:

(1) Order of JOIN: For example: table A and B locate in the same site. They are supposed to join at the site first then sent to the other site.

(2) Access Method: For the join attributes, they can be sorted in some expected order before join. This could reduce some scan time at local site.

(3) Method of JOIN: They used merge JOIN against nested loop JOIN method.

(4) JOIN site: For the two tables at different sites, they have three ways to perform JOIN operation:

- Send outer table to the site of inner table.
• Send inner table to the site of outer table.

• Send both to the third site.

(5) Inner-Table Transfer Strategy: For the inner table being sent to other site, there are two ways to retrieve tuples.

• Re-fetching the inner-table tuples from its store site and send the tuples to other site. This execution need additional communication cost.

• Fetching the whole table and storing it at join site. This execution need additional I/O cost.

2.2.2.2 Dynamic Join strategies with a tree model

The algorithm [22] uses binary join tree to generate join sequence. And it employs dynamic programming.

The distributed optimization process:

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

(2) Chose a optimal site:

The optimal site for materializing is chosen for each relation to be retrieved. The relations may exist in multiple copies in the system or may be fragmented horizontally or vertically.

(3) Data Distribution:

To determine the optimal location of binary join executions and to store intermediate result relations in the available sites.
A binary query tree (BQT) is a query join tree where non-leaf nodes have exactly two sub-nodes (Figure 6).

![Figure 6: A tree model for join strategies](image)

2.2.2.3 Adaptive Join Algorithm for Dynamic Distributed Database

An adaptive query optimization algorithm is proposed in the paper [50]. Its cost model is:

\[
\text{Cost} = \text{optimization overhead} + \text{execution cost}
\]

Both local cost and transmission cost are considered in the paper [50]. The algorithm is composed of the following three phases:

1) Semijoin Phase: in the phase, semijoin is used to reduce the transmission cost.
2) Data Transfer Phase: Joins are employed in the phase to reduce communication cost.
3) Joining Phase: Join is performed after transfer relations to final site.

Instead of a full access plan, the algorithm creates a partial execution plan. Then while updating the statistics, new partial plan is constructed.

### 2.3 Semijoin operation and its application

2.3.1 Semijoin operation
Semijoin [10, 15, 35, 35, 38, 41, 42]: given two relations Ri and Rj. Semijoin is denoted as Ri \bowtie_j R j, where i \neq j, k is common join attribute. The semijoin consists of the following steps.

1. Project relation R_i over join attribute k to get the projection p_{\alpha_i} from site i.
2. Ship the projection p_{\alpha_i} to the site j.
3. Execute p_{\alpha_i} \bowtie_j R_j to get a smaller relation.

![Figure 7: Semijoin cycle](image)

We give an example (Figure 7) with two relations, R1 and R2. Project relation R1 over join attribute k to get the projection p_{\alpha_i} from site 1. Then transfer p_{\alpha_i} to site 2. Execute p_{\alpha_i} \bowtie_j R2 to get a smaller relation (4). Ship (4) in site 2 to R1 to execute join. Since size of relation in (4) is smaller than the relation R2, the transmission cost is
reduced. The goal of semijoin is to obtain the net benefit called cost-effective [40]. The semijoin cost is the size of semijoin projection $p_a$. And the semijoin benefit is the amount of size reduction on relation $R_i$. The semijoin is cost-effective if and only if its benefit is greater its cost.

2.3.2 Application of Semijoin in Distributed DBMS

2.3.2.1 SDD-1

SDD-1[3] is a distributed database system developed by the Computer Corporation of America. It is the first system to allow a relational database to be distributed on a computer network. They used a high level procedural language "Datalanguage" to process queries.

SDD-1 transforms a query into a set of relational calculus expression, which specified data need for the query. Using this set of expressions, a greedy algorithm is used to derive the sequence of semijoins that will retrieve the set of data needed for the query. A major limitation of this algorithm is that it may produce sub-optimal strategies due to the failure to consider other semijoins at each step of the strategy generation. It only considers the data transmission time. The query optimization is aimed to minimize the size of data transferred. The SDD-1 algorithm always selects the most profitable reduction that is immediately at hand.

2.3.2.2 The Algorithm AHY(GENERAL)

The algorithm[1] was developed by Apers, Hevner and Yao. It is based on semijoins. And it aims to minimize either response time or the data transmission time.
(1) Three versions to process complex queries are given.

Version 1: Response Time Version:

Let the transmission cost of the data is the same between any two computers and be a linear function of the size of the data. The function is $C(X) = C_0 + C_1X$ where $X$ is the amount of data transmitted. In the paper [1], assume $C_0 = 0$ and $C_1 = 1$;

For the version, algorithm PARALLEL is applied to minimize response time.

Version 2: Total time version:

Total time of a schedule is the sum of the costs of all transmissions required in the schedule.

An algorithm called SERIAL is applied to minimize the total time.

Version 3: “Handling Redundant Data Transmissions”

(2) Algorithm GENERAL [1] include four phases:

i. Initial all local processing.

ii. For each relation, calculate its schedule.

iii. Integrate the schedules for total time minimization.

iv. Remove those relation schedules which have been transmitted.

2.3.2.3 Two – Way Semijoins

The optimizer [20] reduces transmission costs across a network in distributed query computation. And it has more reduction power than the semijoin.
Two-Way Semijoins Algorithm (Figure 8):

For example: Given relations Ri and Rj which are from site i and j respectively. B is the common attribute.

i. Project relation Ri on attribute B and get Ri[B] which is (B1, B2, B4)

ii. Send Ri[B] from site i to j.

iii. Reduce Rj[B] by eliminating tuples whose attribute B are not matching any of Ri[B]. During reduction of Rj, partition Ri[B] into Ri[B]m (B1, B2) and Ri[A]nm (B4) where Ri[B]m is the set of values in Ri[A] that match one of Rj[A] and Ri[A]nm is equal to Ri[B] – Ri[B]m.

Figure 8: Two-Way semijoin
iv. Send either R[i][B]m or R[B]nm which is smaller to site i. So we send R[B]nm (B4) back.

v. Reduce R[i] with B4 by deleting the tuple matching B4 on attribute B.

2.3.2.4 Domain-Specific Semijoin

For the fragmented database system, semijoin [11] can be performed between relation and relation or relation and fragment. However, it cannot be performed between two fragments, because it causes the elimination of contributive tuples.

The algorithm [11] proposed two modes of executing the semijoin which is the local and the remote domain domain-specific semijoin. They are used in fragment-to-fragment and can provide more flexible distributed query processing while without the loss of contributive tuples.

2.4 Combination of Join and Semijoin and its application

2.4.1 Combination of join and semi-join operations [13, 17] can be used to minimize data transmission cost required for distributed query processing.

Two important concepts are identified in this approach[17]:

(1) Gainful semi-joins: the semi-joins that, even though not profitable themselves, may benefit the execution of subsequent join operations, and become profitable to the use of join operations as reducers.

(2) Pure join attributes: Join attributes that are not part of the output attributes.

The processing plan includes three phase:
(1) "Local processing phase": all processing operations such as selections and projections are processed in local.

(2) "Reduction phase": a semi-join is used to reduce the total communication costs.

(3) "Final processing phase": send all resulting relations to the final site.

An important objective in distributed query processing is to reduce the amount of data transmission required for phase 2 and phase 3.

Semi-joins which become profitable according to the use of join reducers in the paper[13, 17] are termed gainful semi-joins. A semi-join may become gainful due to subsequent join operations. Two non-profitable semi-joins to the same receiving relation cannot be combined to form a pair of profitable semi-joins to that relation. Two non-gainful semi-joins may be combined to form a pair of gainful semi-joins.

A gainful semi-join is different from profitable semi-join that can be evaluated individually.

Chen and Yu[17] use their algorithm which is based on A*[44] algorithm to set the sequence of joins and semijoins.

2.4.2 Chen and Yu proposed an efficient heuristic approach to determine an effective sequence of semijoins and join reducers[16]. Semijoins, whose execution will reduce the amount of data transmission required to perform a join sequence, are termed as benificial semijoins[17] for join sequence. Beneficial semijoins include the conventional profitable semijoins and the gainful semijoins that are not profitable themselves but become beneficial due to the inclusion of join reducers.
Interleaving a join sequence with beneficial semijoin involve following five steps:

1) To obtain a join reducer sequence
2) To Map the join reducer sequence into a join sequence tree.
3) To derive the set of reducible relations for each semijoins.
4) To identify the beneficial semijoin based on the properties of beneficial semijoin developed. Here join sequence tree provides an efficient way to identify for each semijoin its correlated semijoins as well as its reducible relations under the join sequence.
5) To determine the proper order in the combined reducer sequence.

In this way, heuristic helps to reduce the data transfer by determining an effective sequence of join and semijoin reducers. This issue was quite complex as gainful semijoins depend upon subsequent join and semijoin operations. They can not be determined in isolation as profitable semijoins. Obtaining sequence of join reducers involve the method to estimate the effect of set of join operation on query graph. For example, calculation of expected number of tuples after join and determination of pure join attributes and gainful semijoins. Some join attribute which are not the part of final answer is called pure join attribute.

Results in the paper[16] show that the approach of interleaving a join sequence with beneficial semijoins is not only efficient but also effective in reducing the total amount of data transmission required to process distributed queries.

2.5 Bloom filter based Semijoin in Distributed DBMS

Bloom filter was invented by Burton Bloom in 1970[4, 6, 9, 27, 34, 40]. The filter contains an array of bits. It used to reduce accesses to the different files.

The idea (illustrated in Figure 9) is to allocate filter \( v \) with \( m \) bits, initially all set to 0. Let \( A \) be a set of value, and then choose a hash function \( H \). For \( \forall a \in A \), the bit at positions \( H(a) \) in filter \( v \) is set to 1. (A particular bit might be set to 1
multiple times for different element. It is called collision). Given a value $b$ we check the bit at positions $H(b)$. If it is 0 or out of range, then certainly $b$ is not in the set $A$.

Mullin [34] introduced the bloom filter into semijoin in Distributed DBMS. The filter can be used to significantly lower data transmission and local processing costs.

2.5.1 Operation [27]

Let relations $R_i$ and $R_j$ (Figure 9) be two join relations in different location $i$ and $j$. The join attribute is ‘A’.

![Filter v]

$$H(a) = p$$

Figure 9: A bloom Filter with one hash function

(1) Create a filter in site $i$

- Create an array to hold bits.
- Let all bits in the array to zero.
- Developed a hash function $H$. 

22
• For each value of join attribute of \( R_i \), hash on the value(a) and produce an address. Such as \( H(a) = p \).

• Set the value in the address ‘p’ to 1.

(2) Reduce relation size.

• Send the above filter to site \( j \).

• Each value of join attribute of \( R_j \), hashes on the value to produce an address.

• For each above address in the filter, evaluate its value with bitwise operation.

If the value is 1, the corresponding tuple is kept for further processing,

Else if the value is 0, discard the tuple.

Then we send the reduced relation to final sites for Join. Since the size of filters is much smaller than semijoin projection. As a matter of fact, it is more efficient than semijoin. The algorithm is profitable.

2.5.2 Hash-Semijoins

Judy C.R. Chen and Arbee L.P. Chen proposed a relational operator – hash-semijoin. In the paper [40], they transform a simple semijoin into hash-semijoin by replacing traditional semijoin. A full replacement algorithm is given. Comparison for cost-effective is presented by mathematical approach. The result shows that hash-semijoin is more cost-effective than traditional semijoin.

2.5.3 Algorithm X

J.M. Morrissey and Ma, Xiaobo present algorithm X [23]. The algorithm X is a static heuristic for processing general query. Bloom filter techniques is used
instead of traditional semijoin. And both the response time and local processing cost have been reduced.

They compare algorithm X against AHY. The benefits of algorithm X is remarkable. However, algorithm X is based on the perfect hash function.

2.5.4 J.M. Morrissey and W.K. Osborn proposed an algorithm [32] which successfully apply bloom filter over semijoin. And the algorithm can be used to reduce both the response time and local processing cost. A filter rule is applied during reduction. Their evaluation is against full reduction. But the algorithm is also based on perfect hash function. Collision runs [32] are used in their experiments. However, only one hash function are employed for each filter.

The reduction approach [32]

According to the algorithm [32], the execution plan is composed of two phases: Construction of reduction filter and Processing of Queue.

Phase 1: Construction of reduction filter:

1) Create Bloom Filters as reducers. For each common attribute, one filter is created.
   This is first level of construction. With the filter one independent Hash Function is generated.

2) Start to construct filter from the site with lowest number of common attribute.

3) Then reduce the number of common attribute associated by relations.

4) Find the site with lowest number of common attributes except those have processed.
   Process the relation. On the site if a constructed filter size is reduced, and the associated relation currently processing is not in the queue, and they are not the currently processed, put the related relation into a queue for later use.

5) Repeat steps 2), 3) and 4) until all relations have been processed.
Phase 2: Processing of Queue:

In the second level of construction of reduction filter. The actual value in the filter is reduced. In the phase, process only the relations in queue. According to the sequence of queue, first come first process. Repeat steps in phase 1 until the queue is empty. After successful creating reduction filters, use them to reduce relations. Then send those relation to final site for join. This algorithm achieves significantly reducing relation sizes.
CHAPTER 3 ASSUMPTIONS

A relational DBMS must include methods, or algorithms, for implementing the
types of relational operation that can appear in a query execution strategy in
order to perform query optimization. These include the basic relational algebraic
operations, and combinations of these operations. In addition, the DBMS must
have available methods for processing special operations such as aggregation
functions and groupings.

Assumptions

(1) Assume Distributed Database Management System is provided. Data is
distributed on point to point (PPP) network.

(2) There is no fragmentation or replication in the database.

(3) Only Select-Project-Join(SPJ) query is considered. There is no set operations
like UNION, INTERSECTION, PRODUCT, DIFFERENCE involved in the
research.

(4) Non-perfect-Multi-filters could be used for all relations. Collision are
allowed.

(5) The number of hash functions used in the research is less than 9 and none of
them is perfect hash function.

(6) All value type for either experiment or example is integer. The above
assumptions are basis of our approach. The details of our reduction approach
will be introduced in next chapter.
Chapter 4  Introduction of our reduction approach

We employ non-perfect-multi-bloom filters based on the algorithm [32]. Our goal is to provide a practical reduction approach to minimize both data transmission time and local processing time for general queries. The algorithm is heuristic. Our reduction approach follows the algorithm [32] by replacing single perfect hashing on each join attribute based on the approach with multi-filters by non perfect hash function. Since multiple filters is applied to each join attribute, the programming is more complicated. Therefore, some relevant concepts have to be introduced before implementation. Estimation is against the approach [32]. Our reduction approach achieves significantly lowering the cost of transmission. For our reducer, for whole experiments, we can compare with how the number of filters affecting the performance of querying optimization to that generated by the approach in paper [32].

4.1 The difference between the two reduction approaches

1) We use one non-perfect hash function for each filter. As the matter of fact, collisions after hashing on data are not avoidable. However, another approach [32] uses only perfect hash function so that there is no collision.

2) The filter size in reducer [32] must be as large as required by data domain. Our filter size can be controlled. But we use more filters as trade off.

3) In order to reduce collision, we employ multiple filters(Figure 10) on each join attribute instead of using one perfect hashing function on each common join attribute.
4.2 How does our reducer work?

The purpose of our reducer is to apply multi-bloom filters to reduce collision. The reduction quality is improved.

Assume variable a, b, c are unequal integers. And a, b are the only value of join attribute in table A. And c is the only value of same join attribute in table B. The sole tuple c can be kept by our reducer if and only if its values in the all addresses, which are produced by all hashing on c, are 1.

If we apply just one hashing function H1, there is a collision generated by hash function H1 which is H1(a) = H1(c). Therefore, the tuple in table B is remained. If we apply two filters. Then the tuple in table B is thrown because value in address H2(c) is 0. The result is the same as that of reducer [32]. From the example, the conclusion is that the more filters we use, the more amount of tuples for final JOIN can be reduced.

Figure 10: Example for the use of multi-filters
Since using perfect hash function to generate bloom-filter is not realistic in network systems, we propose the reduction approach of multi-bloom filters as a reducer. Obviously, the local cost is increased as trade off. However, our goal is to considerably reduce transmission cost. We will give the experimental comparison results to show how close to the approach [32] according to the number of filters.

4.3 Some related concepts

We provide an example to introduce the use of multi-filters in query processing.

\[\begin{array}{c|c|c}
R1 & A & H \\
\hline
1 & 4 & \\
2 & 2 & \\
3 & 5 & \\
\end{array}\]

\[\begin{array}{c|c|c}
R2 & E & I \\
\hline
3 & 1 & \\
4 & 2 & \\
\end{array}\]

\[\begin{array}{c|c|c|c|c|c|c}
R3 & A & B & C & D & E & F \\
\hline
1 & 4 & 7 & 3 & 6 & & \\
2 & 5 & 10 & 5 & 8 & & \\
10 & 7 & 9 & 7 & 10 & & \\
\end{array}\]

\[\begin{array}{c|c|c|c}
R4 & D & F & G \\
\hline
3 & 6 & 11 & \\
5 & 8 & 12 & \\
5 & 5 & 5 & \\
\end{array}\]

\[\begin{array}{c|c|c|c|c}
R5 & B & C & E & G \\
\hline
4 & 7 & 3 & 11 & \\
8 & 2 & 5 & 12 & \\
9 & 5 & 7 & 13 & \\
\end{array}\]

**Figure 11: Example Relations**

In Figure 11, five relations R1, R2, .. R5 are employed. We use integer as value of our example relations for simplifying our calculation. The capital letters A, B, .., G represent name of attributes.
4.3.1 **Query graph**: Let \( G = (V, E) \) be a query graph. \( G_{\beta} = (V_{\beta}, E_{\beta}) \) is a connected subgraph of \( G \). Let \( R1, R2, \ldots, R5 \) be relationships corresponding to nodes in \( V_{\beta} \) and let \( A, B, C, \ldots, G \) be the distinct attributes associated with edges in \( E_{\beta} \). According to figure 11, a query graph is drawn as figure 12.

4.3.2 **Adjacency Matrix**: It is implemented as a two-dimensional array to describe the relationship between relations (Figure 11) and attributes. "X" means for the relation in the same horizontal line contains the attribute in vertical line. Blank means the relation does not include the attribute. Horizontal line means which attribute the relation owns. Vertical line represents which relation associates with the attribute. If there are more one "X", this attribute regards as join attribute. The degree of a relation is equal to number of join attribute which the node has.
### Table 1: Adjacency Matrix

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R5</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.3.3 "Adjacency List": the list is constructed from Adjacency Matrix. It is a link list which uses to describe the relationship among any relation to all others.

The format of adjacency list:

(a) For first node, the first element is number of relations associated. And the second element is the relation name.

(b) For the following nodes, the first element is the name of associated relation and the second element is the join attribute name.
4.3.4 "Inverted List": It is constructed from Adjacency Matrix. The list represents the relationship among each common join attribute and the relations with the attributes. Order is not important.

```
A → R1 → R3
B → R3 → R5
C → R3 → R5
D → R3 → R4
E → R2 → R5
F → R3 → R4
G → R4 → R5
```

4.3.5 "Queue": A link list to hold relation names. The queue follows "first come, first service" rule.

4.3.6 Filter Rule: Use "filter rule" to decide which relations should be added into Queue.

(1) If any filter for a specific join attribute of relation $R_i$ is changed.

(2) And if it is not in the Queue.

(3) And if it is not the relation which is currently processed.
If the three conditions are satisfied, the relation can be added to the Queue.

4.3.7 Filter List: A list which used to indicate filters already available for common attributes.

4.4 Description of the use of multi-bloom filters based reduction approach

**Phase I: construction of reduction filters**

Steps:

1. Select the relation with lowest in-degree. If more than one relations have the lowest in-degree, we select a relation depending on the order of "Adjacency List".

2. Search filter list.

   - If no corresponding filter available, produce filters with the join attribute of the relation and put all non-duplicate value in the filter. Then add the filter into "Filter List". In this case, no filter goes to "Queue".

   - Else if corresponding filters already exist, reduce the relation with the filters.

The reduction rule:

For each value of the join attribute of the relation, apply all hash functions to it and produce all address from filters. Only when all value in those addresses are 1. we keep the tuple for further processing. Otherwise, delete the tuple.
- Then generate temporary filters. Compare with sizes of the temporary filters to their previous sizes. If the size of temporary filters is less, we will use the temporary filters later. Regarding to "Inverted List" and "Filter Rule", place the relation into "Queue".

(3) Reduce in-degree of relation in "Adjacency List".

(4) Mark(-1) the relation as processed.

(5) Repeat until all relations in "Adjacency List" have been processed.

**Phase II: Processing of Queue**

Use "Queue" to decide which relations must be further produced.

Steps:

(1) Take relation off from the "Queue" in its order.

(2) Reduce relation using filters.

(3) If filter is changed, use "Filter Rule".

(4) Repeat until "Queue" is empty

### 4.5 Construct efficient hash functions

Because the non-perfect-hash functions are applied for the research, collisions are not avoidable. We aim to construct efficient hash functions to reduce collisions as many as possible. If collisions are generated, then we apply another bloom-filter. We can reduce the collision of those values which cause previous collision. If the more hash functions are applied, the more collisions can be minimized.
The new algorithm for hash function is based on the popular method: middle-square method[46].

Theorem 1: let m be the filter size, p is an integer $0 \leq p \leq m$ and p is the largest primary integer. Variable $\alpha$ is an integer.

i. Let $\beta = \alpha + n$, which n is a factor integer and $n>100$.

ii. Square $\beta$, let $\gamma = \beta \times \beta$.

iii. Starting from second right most digit of $\gamma$, extract three digits and generate an integer denoted as ‘i’.

iv. Hash function is: $H(a) = \text{"i Moded by p"}$. Variable a is an integer.

4.6 An example

The following is an example used to describe our reduction approach. It gives us an profile for our approach.

4.6.1 Assumption:

(1) The hash function introduced in 4.5 will be used to in the example.

Only two hash functions are applied. According to Theorem 1, let $n = 111$ and $n= 115$ respectably.

One hash function is denoted as H1 and another is denoted as H2.

(2) Filter size is 3 which means we only use three bits.

(3) For any specific attribute x, Filter 1, denote as Fx1 and Filter 2, denote as Fx2.
4.6.2 Details of the example

**Phase I: Construction of reduction filters**

(1) R1 and R2 have the lowest indegree. We select R1, since R1 is prior to R2 in Adjacency List.

The join attribute for R1 is A. Obviously there is nothing in Filter list so that we just simply construct two filters with two hash functions, then put A into Filter List. The result in digit is A: 1,2,3

For Join Attribute A:

\[ H1(1) = 2; \quad H2(1) = 0; \]
\[ H1(2) = 0; \quad H2(2) = 2; \]
\[ H1(3) = 2; \quad H2(3) = 2; \]

\[ F_{A1} : \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \quad F_{A2} : \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \]

Since the new filters for attribute A is not updated, no relation will be put on the Queue. The vertex R1 and the edge for attribute A are removed from the query graph(Figure 14).

![Query graph diagram](image)

Figure 14: Query graph After delete node R1 and edge A
Adjacency List is updated as the followings:

\[-1 \quad R1 \quad R3 \quad A\]

\[4 \quad R3 \quad R1 \quad A \quad R5 \quad B \quad R5 \quad C \quad R4 \quad D \quad R4 \quad F\]

Construct Filter List: A

(2) Select R2 since it is the only one which has the lowest indegree. The join attribute for R1 is E. There is no attribute E in Filter list so that we just simply construct filters, then put E into Filter List. The result in digit is E: 3, 4

For Join Attribute E:

\[H1(3) = 2; \quad H2(3) = 2;\]

\[H1(4) = 1; \quad H2(4) = 2;\]

\[\begin{array}{ccc}
F_{e1}: & 0 & 1 & 1 \\
F_{e2}: & 0 & 0 & 1
\end{array}\]

Since the new Filters for Attribute A is not updated, no relation will be put on the Queue. The vertex R1 and the edge for attribute A are removed (Fig. 11) from the query graph.
Figure 15: Query graph After delete node R2 and edge E

Adjacency List is updated:

Three edges are updated:

-1 R2 \rightarrow R5 E

3 R5 \rightarrow R3 B \rightarrow R3 C \rightarrow R2 E \rightarrow R4 G

Update Filter List: A, E

(3) So far, the lowest indegree is 3. Both R4 and R5 have the lowest indegree. We select R4, since R4 is prior to R5 in Adjacency List.

Similarly, there is no attribute D, F and G in the Filter List. We construct filters for the attributes D, F and G, then put them into Filter List. The results in digit are:

D: 3, 5
F: 6, 8, 5
G: 11, 12, 5

For Join Attribute D:

H1(3) = 2; H2(3) = 2;
H1(5) = 0;    H2(5) = 2;

F_{D1}:
\[
\begin{array}{ccc}
1 & 0 & 1 \\
\end{array}
\]
F_{D2}:
\[
\begin{array}{ccc}
0 & 0 & 1 \\
\end{array}
\]

For Join Attribute F:

H1(6) = 2;    H2(6) = 2;
H1(8) = 0;    H2(8) = 2;
H1(8) = 0;    H2(5) = 2;

F_{F1}:
\[
\begin{array}{ccc}
1 & 0 & 1 \\
\end{array}
\]
F_{F2}:
\[
\begin{array}{ccc}
0 & 0 & 1 \\
\end{array}
\]

For Join Attribute G:

H1(11) = 2;    H2(11) = 2;
H1(12) = 0;    H2(12) = 0;
H1(5) = 0;    H2(5) = 2;

F_{G1}:
\[
\begin{array}{ccc}
1 & 0 & 1 \\
\end{array}
\]
F_{G2}:
\[
\begin{array}{ccc}
1 & 0 & 1 \\
\end{array}
\]

Since the new filters for Attribute D, F and G are not updated, no relation will be put on the Queue. The vertex R4 and the edge for attributes D, E and G are removed from the query graph (Figure 15).
Figure 16: Query graph After delete note R4 and edge D, F and G

Adjacency List is updated:

Update Filter List: A,D,E,F and G

(4) Both R3 and R5 have the lowest indegree 2. We select R3, since R3 is prior to R5 in Adjacency List. The relation contains join attributes A, B, C, D and F.

The Filter List already contains attributes A, D and F.

i. Apply reduction filters for attribute A to relation R3.

The value in attribute A of R3 is: 1, 2, 10

The first two tuples will not be changed.

For 10, \( H1(10) = 2, H2(10) = 1 \). Therefore the tuple will be removed.
The resulting table is:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>

ii. Apply reduction filters for attribute D and F to relation R3. There is no change for the resulting relation.

iii. Since filter F has been changed. Update Queue with inverted List and "Filter Rule".

We add relation R4 into Queue

iv. For Join Attribute B. Th Filter for B in digit is: 4, 5

   H1(4) = 1;   H2(4) = 2;

   H1(5) = 0;   H2(5) = 2;

   \[ F_{b1}: \begin{array}{ccc} 1 & 1 & 0 \end{array} \quad F_{b2}: \begin{array}{ccc} 0 & 0 & 1 \end{array} \]

For Join Attribute C, Th Filter for C in digit is: 7, 10

   H1(7) = 1;  H2(7) = 2;

   H1(10) = 0; H2(10) = 2;

   \[ F_{c1}: \begin{array}{ccc} 0 & 0 & 1 \end{array} \quad F_{c2}: \begin{array}{ccc} 0 & 1 & 1 \end{array} \]
Because the new filters for attribute A, B, C and D are not updated, they will not be put on the Queue. But filter for attribute F is changed. Apply reduction filters for attribute F.

Update Queue: R4

Updated filters for attribute F:

\[
F_{F_1} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \quad F_{F_2} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}
\]

The vertex R3 and the edge for attributes B, C are removed (Fig.) from the query graph. We have the following query graph.

![Query Graph](image)

Figure 17: Query graph After delete note R4 and edge D, F and G

Adjacency List is updated:

-1 R3 → R1 A → R5 B → R5 C → R4 D → R4 F

0 R5 → R3 B → R3 C → R2 E → R4 G

Update Filter List: A, B, C, D, E, F and G

(5) Only one node R5 is left with indegree 0. We select the relation R5. The relation contains join attributes B, C, E and G.

The Filter List already contains all join attributes.
a) Apply reduction filters for attribute B to relation R5.

The value in attribute A of R3 is: 4, 8, 9

The first tuple will be kept.

For 8, \( H1(8) = 2 \) (but we do not have to calculate \( H2(8) = 2 \), since in \( F \) the value on address 2 is 0). Therefore the tuple will be removed.

For 9, \( H1(9) = 1 \), \( H2(8) = 2 \). Therefore the tuple will be removed.

The resulting table is:

\[
\begin{array}{cccc}
R5 & B & C & E & G \\
\hline
4 & 7 & 3 & 11 \\
\end{array}
\]

b) There are no change for join attributes C, E and G for resulting relation.

c) The filter for join attributes B, C, E and G have been changed.


Update Queue: R4 -> R3 -> R2

Th Filter for B in digit is: 4

Th Filter for C in digit is: 7

Th Filter for E in digit is: 3

Th Filter for G in digit is: 11

Update filters:
Adjacency List is updated:

\[
\begin{array}{ccc}
F_{S1} & F_{E1} & F_{G1} \\
0 & 0 & 1 \\
F_{C1} & F_{E2} & F_{G2} \\
0 & 0 & 1 \\
\end{array}
\]

**Phase II: Processing of Queue**

The Queue is R4 -> R3 -> R2

(1) Select relation R4.

Apply reduction filters for attribute D, F and G to relation R4. No filter are changed. Therefore no relation will go to the Queue. And R4 is removed from the Queue.

The resulting relation:

<table>
<thead>
<tr>
<th>R4</th>
<th>D</th>
<th>F</th>
<th>G</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>
(2) Select relation R3

Apply reduction filters for attribute a, B, C, D and F to relation R3. The filters for attributes B, C, D, F are not changed. But filters for attribute A has been changed. Update Queue with inverted List and “Filter Rule”.

We add relation R1 into Queue. And remove R3 from the Queue.

The resulting relation:

<table>
<thead>
<tr>
<th>R3</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

The Queue is: R2 -> R1

(3) Select relation R2.

- Apply reduction filters for attribute E to relation R2. No filter are changed. Therefore no relation will go to the Queue. And R4 is removed from the Queue.

- The resulting relation:

<table>
<thead>
<tr>
<th>R2</th>
<th>E</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

(4) Select relation R1.
• Apply reduction filters for attribute A to relation R1. No filter are changed. Therefore no relation will go to the Queue. And R1 is removed from the Queue.

• The resulting relation:

\[
\begin{array}{cc}
R1 & \text{A} & \text{H} \\
1 & 4
\end{array}
\]

(5) The Queue is empty. All relations in the Queue have been processed.

\[
\begin{array}{cccc}
\text{R1} & \text{A} & \text{H} & \text{R2} & \text{E} & \text{I} & \text{R3} & \text{A} & \text{B} & \text{C} & \text{D} & \text{F} \\
1 & 4 & \text{R4} & \text{D} & \text{F} & \text{G} & \text{R5} & \text{B} & \text{C} & \text{E} & \text{G} \\
3 & 6 & 11 & 4 & 7 & 3 & 11
\end{array}
\]

Figure 18: Reduced relations

\[
\begin{array}{cccccccc}
\text{A} & \text{B} & \text{C} & \text{D} & \text{E} & \text{F} & \text{G} & \text{H} & \text{I} \\
1 & 4 & 7 & 3 & 3 & 6 & 11 & 1 & 1
\end{array}
\]

Figure 19: Result of joining the reduced relations
4.7 Summary

Our new reduction approach which using multi-bloom filters with illustrative example is proposed in the chapter. We aimed to minimize data transmission time. The followings will perform experiments according above reduction approach.
In this chapter, the methodology for our experiments is introduced. Since the performance of our reducer will not compare with any other algorithm and it is hard to use mathematical method to deduct the evaluation, the experimental evaluation is the most efficient technique for our research. The evaluation is against the reduction approach [32].

5.1 Methodology

Some relevant issues about how to evaluate our experiments are described in the chapter. Our evaluation is against the reducer [32].

5.1.1 The factors affecting our evaluation

For the general queries we defined in Chapter 3, the possible factors affecting our evaluation are described as following:

(1) Query type: denoted as “n-m”, while ‘n’ indicates the number of relations involved in the query and ‘m’ indicates the number of join attributes.

(2) Hash function: the function mapping an attribute value to an address.

(3) Number of hash functions: the number of hash functions which are applied to single bloom-filter.

(4) Domain size of filter: number of total bits in a filter.
(5) Filter size: total number of bits whose value is 1.

(6) Selectivity: For a join attribute of a relation, selectivity is equal to the number of distinct attribute value divided by the attribute domain size. If its ratio is low, we consider selectivity is high. Otherwise, we consider it is low.

(7) Connectivity: The total number of existing join attributes in the query divided by the total number of possible join attribute (denoted as n*m).

(8) Attribute domain size: the number of distinct attribute value which may appear in an attribute.

(9) Relation cardinality: number of tuple of a relation.

5.1.2 Evaluating experimental results

Our comparison is against the reduction approach [32]. In our experiments, we develop two new methods to evaluate the performance of our reducer for our experiments.

**Definition 1**: Extra percentage reduction data size of our reducer against the reducer [32], denoted as Ex_P.

Assume a set of queries which total number is n. For each query i(i is an integer), let Xi, Yi be size of final reduction produced by our reducer and the reducer [32] respectively.

Then \[ \text{Ex}_P = \left( \frac{\sum_{i=1}^{n} X_i}{\sum_{i=1}^{n} Y_i} - 1 \right) \times 100\% \]
If Ex-P is close to 0%, that means the performance of our reducer is close to that of the reducer [32].

**Definition 2**: Percentage of fully reduced queries against the those of reducer [32], denoted as Ful_P.

Assume a set of queries which total number is n. For each query i (i is an integer), let X, Y be total number of final fully reduced queries produced by our reducer and the reducer [32] respectively.

Then \[ \text{Ful}_P = \frac{X}{Y} \times 100\% \]

If Ex-P is close to 100%, that means the performance of our reducer is close to that of the reducer [32]. This percentage will help us find out the capability of our reducer achieving fully reduction against that of the reducer [32] doing

### 5.2 Experimental plan

1. Query type: vary from 3-2 to 6-4.
2. Hash function: defined in Chapter 4.
3. Number of hash functions: varying from 1 to 9.
4. Domain size of filter: we set its size as one and half of common join attribute domain size.
5. Selectivity: very from 0.02-0.4, 0.4-0.7 and 0.7-0.95.
(6) Connectivity: 75% or 100%.

(7) Attribute domain size: 150-250.

(8) Relation cardinality: 200-600.

The experiment will be divided into two groups. One is regarding connectivity 75%, while another is 100%. For each query we apply one perfect hash function. Meanwhile we use different number of non-perfect on the query for comparison. The experiments applied more than 20,000 queries for total. And 1,800 queries for each type.

This chapter provide us frame work of experiments. In next chapter, we analyzes the experiment results.
CHAPTER 6  ANALYSIS OF EXPERIMENTAL RESULTS

For the purpose of showing how close our reduction performance to that of the reducer in [32]. Simulating the reduction approach [32] to apply perfect hash function for each bloom-filter to get the size of reduced relation. In order to guarantee our experiment is on right basis, we simulate the reduction approach [32] to perform experiments. Meanwhile, for each query, we also apply our reduction approach with different number of bloom-filters. We calculate all the reduction data sizes with different reduction approach: fully reduced approach, the reduction approach [32] and our approach. Then we generate Ex_P and Ful_P tables. After testing experiments, we found that after we apply six or more filters Ex_P is 0% and Ful_P achieves 100%. Therefore, we can just list at most five filters in our experimental tables.

Details of experimental results and analysis

6.1 Presents the experimental result which had been made with reducer [32].

1. In order to compare the performance with the reducer [32]. For each query in our experiments, we perform the experiments which were done in [32]. Finally, we found that the results are almost the same as that in [32]. This can guarantee our comparison is correct. The evaluation is against full reducer which can fully reduces relations to those tuples which only participate in final joins. In table 2, “Ex” is the extra percentage reduction sizes produced
by the reducer [32] against full reduction approach. And “Ful” means that the percentage of queries which achieve full reduction [32].

<table>
<thead>
<tr>
<th>Type</th>
<th>Connectivity = 75%</th>
<th></th>
<th>Connectivity = 100%</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Selectivity 0.02-0.4</td>
<td>Selectivity 0.4-0.7</td>
<td>Selectivity 0.7-0.95</td>
<td>Selectivity 0.02-0.4</td>
</tr>
<tr>
<td></td>
<td>Ex</td>
<td>Ful</td>
<td>Ex</td>
<td>Ful</td>
</tr>
<tr>
<td>3-2</td>
<td>41</td>
<td>93.2</td>
<td>5.06</td>
<td>60.98</td>
</tr>
<tr>
<td>3-3</td>
<td>0.76</td>
<td>97.92</td>
<td>2.05</td>
<td>88.45</td>
</tr>
<tr>
<td>3-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>4-2</td>
<td>0.46</td>
<td>98.62</td>
<td>2.51</td>
<td>87.76</td>
</tr>
<tr>
<td>4-3</td>
<td>0</td>
<td>100</td>
<td>1.37</td>
<td>78.04</td>
</tr>
<tr>
<td>4-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>5-2</td>
<td>0</td>
<td>100</td>
<td>0.79</td>
<td>77.09</td>
</tr>
<tr>
<td>5-3</td>
<td>0</td>
<td>100</td>
<td>0.77</td>
<td>89.36</td>
</tr>
<tr>
<td>5-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6-2</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6-3</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Avg</td>
<td>0.22</td>
<td>99.15</td>
<td>1.13</td>
<td>90.14</td>
</tr>
</tbody>
</table>

Table 2: The result of applying the reducer [32]

In the table 2, the results are ideal because it is very close to that of full reducer. For more details, please find in the paper [32].

6.2 Experimental results by applying one to six bloom-filters.
For each experiment, we produce the results for both extra percentage size(Ex_P) and the percentage of fully reduced relations(Ful_P) against the reducer[32]. Then we analyze our results through three perspectives – total average, connectivity and selectivity.

6.2.1 The following table shows the actual results which apply one bloom-filter to each common join attribute. The analysis is based on table 3. It includes three aspects:

<table>
<thead>
<tr>
<th></th>
<th>Selectivity 0.02-0.4</th>
<th>Selectivity 0.4-0.7</th>
<th>Selectivity 0.7-0.95</th>
<th>Selectivity 0.02-0.4</th>
<th>Selectivity 0.4-0.7</th>
<th>Selectivity 0.7-0.95</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Ex_P</td>
<td>Ful_P</td>
<td>Ex_P</td>
<td>Ful_P</td>
<td>Ex_P</td>
<td>Ful_P</td>
</tr>
<tr>
<td>3-2</td>
<td>163.64</td>
<td>60</td>
<td>406.39</td>
<td>19.57</td>
<td>408.65</td>
<td>11.43</td>
</tr>
<tr>
<td>3-3</td>
<td>82.22</td>
<td>92.86</td>
<td>109.64</td>
<td>85.39</td>
<td>145.28</td>
<td>78.67</td>
</tr>
<tr>
<td>3-4</td>
<td>75.07</td>
<td>98.99</td>
<td>30.6</td>
<td>99.96</td>
<td>45.18</td>
<td>93.68</td>
</tr>
<tr>
<td>4-2</td>
<td>192.91</td>
<td>90.62</td>
<td>212.5</td>
<td>56.52</td>
<td>313.74</td>
<td>90</td>
</tr>
<tr>
<td>4-3</td>
<td>120.31</td>
<td>97.94</td>
<td>154.65</td>
<td>89.13</td>
<td>190.61</td>
<td>94.1</td>
</tr>
<tr>
<td>4-4</td>
<td>0</td>
<td>100</td>
<td>142.95</td>
<td>98.99</td>
<td>136.54</td>
<td>98.98</td>
</tr>
<tr>
<td>5-2</td>
<td>179.02</td>
<td>94.74</td>
<td>201.84</td>
<td>66.67</td>
<td>508.27</td>
<td>66.67</td>
</tr>
<tr>
<td>5-3</td>
<td>24.95</td>
<td>98.99</td>
<td>89.68</td>
<td>98.97</td>
<td>125.07</td>
<td>94.9</td>
</tr>
<tr>
<td>5-4</td>
<td>0</td>
<td>100</td>
<td>63.24</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6-2</td>
<td>136.97</td>
<td>97.96</td>
<td>272.72</td>
<td>83.87</td>
<td>371.76</td>
<td>72.29</td>
</tr>
<tr>
<td>6-3</td>
<td>8.17</td>
<td>98.99</td>
<td>170.49</td>
<td>98.98</td>
<td>275.23</td>
<td>94.9</td>
</tr>
<tr>
<td>6-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Avg</td>
<td>81.94</td>
<td>94.26</td>
<td>154.56</td>
<td>83.17</td>
<td>218.36</td>
<td>82.97</td>
</tr>
</tbody>
</table>

Table 3: The results of applying one bloom-filter
(1) On average: approximately 95.51% extra size would be produced in the final results. Moreover, for Fulp, about 83.38% relations can achieve fully reduced against the reducer[32]. The worst “Ex_P” is 324.99, which happened with type 3-2, connectivity 100% and selectivity 0.7-0.95. The worst Fulp is 11.43%, which happened with type 3-2, connectivity 75% and selectivity 0.4-0.7. The result shows that if we simply apply one filter, our approach is definitely not ideal. Therefore multi-filters are necessary.

(2) On connectivity: For connectivity 75%, the average Ex_p and Fulp are 151.62% And 86.8% respectively. For connectivity 100%, the average Ex_p and Fulp 46.68% and 90.01% respectively. The results indicate the Ex_p decreased while connection increased. The difference between the two connectivity is 104.94% which is significant improvement. However, For Fulp, there is only 7% difference. The percentage of fully reduced relations is increasing while its connectivity creased.

(3) On Selectivity: The best and worst average Ex_p are {48.89%, 153.15%} when selectivity is 0.02-0.4 and 0.7-0.95. The best and worst average Fulp are, in selectivity 0.02-0.4 and 0.7-0.95, {96.54%, 64.02%}. The results indicate that while the selectivity is decreased the reduction performance goes worse.

6.2.2 The following table shows the actual results of applying two bloom-filters. If two filters are applied the reduction quality is impressively improved.
<table>
<thead>
<tr>
<th>Type</th>
<th>Selectivity</th>
<th>Connectivity = 75%</th>
<th>Connectivity = 100%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Ex_P</td>
<td>Ful_P</td>
</tr>
<tr>
<td>3-2</td>
<td>3.4</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>3-3</td>
<td>0</td>
<td>100</td>
<td>1.6</td>
</tr>
<tr>
<td>3-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>4-2</td>
<td>0</td>
<td>100</td>
<td>0.8</td>
</tr>
<tr>
<td>4-3</td>
<td>0</td>
<td>100</td>
<td>0.28</td>
</tr>
<tr>
<td>4-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>5-2</td>
<td>0</td>
<td>100</td>
<td>1.07</td>
</tr>
<tr>
<td>5-3</td>
<td>0</td>
<td>100</td>
<td>0.47</td>
</tr>
<tr>
<td>5-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>6-2</td>
<td>0</td>
<td>100</td>
<td>1.94</td>
</tr>
<tr>
<td>6-3</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>6-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>Avg</td>
<td>0.03</td>
<td>100</td>
<td>0.67</td>
</tr>
</tbody>
</table>

Table 4: The results of applying two bloom-filters

(1) On total average: Only 1.41% of extra size would be produced in the final results. Moreover, for Ful_P, almost 83.38% of queries generated by our reducer can be fully reduced against the reducer [32]. The worst Ex_P is 19.26%, which happened with type 3-2, connectivity 75% and selectivity 0.7-0.95. The worst Ful_P is 51.43%, which happened with type 3-2, connectivity 75% and selectivity 0.7-0.95. Comparing with the result with that by two filters, there is no wide difference.

56
(2) On connectivity: For connectivity 75%, the average Ex_P and Ful_P are 2.43% and 97.43% respectively. For connectivity 100%, the average Ex_P and Ful_P are 0.38% and 99.18% respectively. The results indicate that the Ex_P decreased while connectivity get increased. The difference of Ex_P between connectivity 75% and 100% is 2.05% which is significant improvement. However, for Ful_P, there is only 1.75% difference. The percentage of full reduced relations is increasing while its connectivity is increased.

(3) On Selectivity: The best and worst average Ex_P are, in selectivity 0.02-0.4 and 0.7-0.95, {0.02%, 3.64%}. The difference is 3.62% which is slight. The best and worst average Ful_P are, in selectivity 0.02-0.4 and 0.7-0.95, {100%, 97.46%}. The difference is 2.54% which is not significant.

6.2.3 The following table shows the actual data of using three bloom-filters.

The analysis is based on table 5. It includes four aspects:

(1) On total average: Approximately 0.23% extra size would be produced in the final results. Furthermore, for Ful_P, about 99.24% of queries generated by our reducer can be fully reduced against the reducer[32]. The worst “Ex_P” is 2.65%, which happened with type 3-2, connectivity 75% and selectivity 0.7-0.95. The worst Ful_P is 57.14%, which happened with type 3-2, connectivity 75% and selectivity 0.4-0.7. The result shows that if we apply three bloom filters, the reduction performance is gradually improved.

(2) On connectivity: For connectivity 75%, the average Ex_P and Ful_P are 0.45% and 98.47% respectively. For connectivity 100%, the average Ex_P
<table>
<thead>
<tr>
<th>Type</th>
<th>Selectivity</th>
<th>Selectivity</th>
<th>Selectivity</th>
<th>Selectivity</th>
<th>Selectivity</th>
<th>Selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ex_P</td>
<td>Ful_P</td>
<td>Ex_P</td>
<td>Ful_P</td>
<td>Ex_P</td>
<td>Ful_P</td>
</tr>
<tr>
<td>3-2</td>
<td>0</td>
<td>100</td>
<td>0.34</td>
<td>95.65</td>
<td>2.65</td>
<td>57.14</td>
</tr>
<tr>
<td>3-3</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>1.96</td>
<td>100</td>
</tr>
<tr>
<td>3-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0.61</td>
<td>100</td>
</tr>
<tr>
<td>4-2</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0.55</td>
<td>92.18</td>
</tr>
<tr>
<td>4-3</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0.39</td>
<td>100</td>
</tr>
<tr>
<td>4-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>3.11</td>
<td>100</td>
</tr>
<tr>
<td>5-2</td>
<td>0</td>
<td>100</td>
<td>0.13</td>
<td>100</td>
<td>6.36</td>
<td>100</td>
</tr>
<tr>
<td>5-3</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0.15</td>
<td>100</td>
</tr>
<tr>
<td>5-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6-2</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0.06</td>
<td>100</td>
</tr>
<tr>
<td>6-3</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Avg</td>
<td>0</td>
<td>100</td>
<td>0.04</td>
<td>99.64</td>
<td>1.32</td>
<td>95.78</td>
</tr>
</tbody>
</table>

Table 5: The results by applying three bloom - filters

and Ful_P 0.01% and 99.99% respectively. The results indicate that the Ex_P decreased when connectivity is increased. The difference of Ex_P between connectivity 75% and 100% is 0.44% which is significant improvement. However, For Ful_P, there is only 1.52% difference. The percentage of fully reduced queries is increasing while its connectivity is increased.

(3) On Selectivity: The best and worst average Ex_P are, in selectivity 0.02-0.4 and 0.7-0.95, {0%, 0.68%}. The difference is 0.68% which differs a litter. The best and
worst average Ful_P are, in selectivity 0.02-0.4 and 0.7-0.95, {100%, 97.89%}. The difference is 2.11% which difference is also small.

6.2.4 The following table shows the actual data of applying four bloom-filters. The analysis is based on Table 6. The results show that the reduction performance is very close to that of the reducer [32].

<table>
<thead>
<tr>
<th>Type</th>
<th>Connectivity = 75%</th>
<th>Connectivity = 100%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Selectivity 0.02-0.4</td>
<td>Selectivity 0.4-0.7</td>
</tr>
<tr>
<td></td>
<td>Ex_P</td>
<td>Ful_P</td>
</tr>
<tr>
<td>3-2</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>3-3</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>3-4</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>4-2</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>4-3</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>4-4</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>5-2</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>5-3</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>5-4</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6-2</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6-3</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6-4</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Avg</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 6: The results of applying four bloom - filters

(1) On total average: Approximately 0.02% extra size would be produced in the final results. And for Ful_P, about 99.51% relations can achieve fully reduced
against the reducer[32]. The worst "Ex_P" is 0.14%, which happened with type 3-2, connectivity 75% and selectivity 0.7-0.95. The worst Ful_P is 82.86%, which happened with type 3-2, connectivity 75% and selectivity 0.4-0.7.

(2) On connectivity: For connectivity 75%, the average Ex_P and Ful_P are 0% and 99.12% respectively. For connectivity 100%, the average Ex_P and Ful_P 0% and 100% respectively. The results indicate that there is almost no extra size produced in our final join relations. However, For Ful_P, there is 0.88% difference.

(3) On Selectivity: The best and worst average Ex_P in selectivity 0.02-0.4 and 0.7-0.95 are {0%, 0.04%} . Their difference is 0.04% which almost can be neglected. The best and worst average Ful_P in selectivity 0.02-0.4 and 0.7-0.95 is {100%, 99.12%}. Their difference is 0.88% which can be ignored.

6.2.5 The table 7 shows the actual data after we apply five bloom-filters.

The following analysis is based on Table 7. It includes three aspects.

(1) On average: approximately 0% extra size would be produced in the final results. For Ful_P, about 99.82% of queries can achieve fully reduced against the reducer [32]. The worst "Ex_P" is 0.05%, which happened with type 3-2, connectivity 75% and selectivity 0.7-0.95. The worst Ful_P is 91.43%, which happened with type 3-2, connectivity 75% and selectivity 0.4-0.7. If four bloom-filters are used, the results are satisfied.
<table>
<thead>
<tr>
<th>Type</th>
<th>Selectivity</th>
<th>Selectivity</th>
<th>Selectivity</th>
<th>Selectivity</th>
<th>Selectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.02-0.4</td>
<td>0.4-0.7</td>
<td>0.7-0.95</td>
<td>0.02-0.4</td>
<td>0.4-0.7</td>
</tr>
<tr>
<td></td>
<td>Ex_P</td>
<td>Ful_P</td>
<td>Ex_P</td>
<td>Ful_P</td>
<td>Ex_P</td>
</tr>
<tr>
<td>3-2</td>
<td>0</td>
<td>100</td>
<td>0.03</td>
<td>97.83</td>
<td>0.05</td>
</tr>
<tr>
<td>3-3</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>3-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>4-2</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0.01</td>
</tr>
<tr>
<td>4-3</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>4-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>5-2</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>5-3</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>5-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>6-2</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0.02</td>
</tr>
<tr>
<td>6-3</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>6-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>Avg</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>99.82</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7: The results of applying five bloom-filters

(2) On connectivity: For connectivity 75%, the average Ex_P and Ful_P are 0% and 99.65% respectively. For connectivity 100%, the average Ex_P and Ful_P are 0% and 100% respectively. The results indicate the Ex_P decreased while connectivity is increased. There is no difference of Ex_P regarding to different connectivities in our experiments. However, for Ful_P, there is only 0.35% difference.
(3) On selectivity: The best and worst average Ex_P in selectivity 0.02-0.4 and 0.7-0.95 is \{0\%, 0\%\}. The first time we found that there is no difference between our reducer and the reducer [32]. The best and worst average Ful_P in selectivity 0.02-0.4 and 0.7-0.95 are \{100\%, 99.56\%\}.

6.2.6 For our all experiments, if and only if applying six or more bloom-filters, every Ex_P and Ful_P can achieve 0\% and 100\% respectively. The results of our approach are exactly the same as the reducer [32] does. Therefore, the results are ideal if we use six bloom-filters. No necessary to use any extra bloom-filters.

6.3 SUMMARY

6.3.1 Comparison of the results according to number of bloom-filters and connectivity. Apparently, according to table 8, the performance of our reducer is improved with the increase of the connectivity.

<table>
<thead>
<tr>
<th>Connectivity</th>
<th>Ex_P</th>
<th>Full_P</th>
<th>Ex_P</th>
<th>Full_P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. Of Filters</td>
<td>75%</td>
<td>100%</td>
<td>75%</td>
<td>100%</td>
</tr>
<tr>
<td>1</td>
<td>151.62</td>
<td>86.8</td>
<td>46.68</td>
<td>90.1</td>
</tr>
<tr>
<td>2</td>
<td>2.42</td>
<td>97.43</td>
<td>0.38</td>
<td>99.18</td>
</tr>
<tr>
<td>3</td>
<td>0.45</td>
<td>98.47</td>
<td>0.01</td>
<td>99.99</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>99.12</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>99.65</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 8. The reduction result of varying from one to six filters and connectivity.
There is dramatic difference of Ex_P which happened with one bloom-filter. The differences of Ex_P between connectivity 75% and 100% are 104.94%, which are much higher average. For those cases which more than one filter is used, there are slight difference with varying of connectivity for both Ex_P and Ful_P.

6.3.2 Comparison of the results according to number of relations and selectivity

<table>
<thead>
<tr>
<th>Selectivity</th>
<th>0.02 - 0.4</th>
<th>0.4 - 0.7</th>
<th>0.7 - 0.95</th>
</tr>
</thead>
<tbody>
<tr>
<td>Num. Of Filters</td>
<td>Ex_P</td>
<td>Full_P</td>
<td>Ex_P</td>
</tr>
<tr>
<td>1</td>
<td>48.89</td>
<td>96.54</td>
<td>85.42</td>
</tr>
<tr>
<td>2</td>
<td>0.02</td>
<td>100</td>
<td>0.57</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>100</td>
<td>0.02</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>100</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 9: Reduction results of Applying from one to six filters with varying selectivity.

The results in table 9 show that the performance of our reducer performs better when the selectivity is increased. There is high difference happened within the result if one bloom-filter is used for Ex_P. However, if we use more than one bloom-filter, there are slight difference of Ex_P with varying of selectivity. From
the average, we found the difference between selectivity 0.02-0.4 with 0.4-0.7 is much smaller than that between selectivity between 0.4-0.7 to 0.95. For those cases which more than one filter is used, there are slight difference with varying of selectivity for both Ex_P and Full_P.

1. Comparison of results with different query types by applying one to six bloom- filters. The analysis concentrates on the variation of query types.

<table>
<thead>
<tr>
<th>No. Filters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ex_P</td>
<td>Ful_P</td>
<td>Ex_P</td>
<td>Ful_P</td>
<td>Ex_P</td>
<td>Ful_P</td>
</tr>
<tr>
<td>3-2</td>
<td>295.1</td>
<td>42.79</td>
<td>5.77</td>
<td>82.60</td>
<td>0.5</td>
<td>92.1</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>96.78</td>
<td>0.02</td>
<td>98.21</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>3-3</td>
<td>56.19</td>
<td>92.82</td>
<td>2.56</td>
<td>99.78</td>
<td>0.38</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>3-4</td>
<td>25.14</td>
<td>98.77</td>
<td>0.82</td>
<td>100</td>
<td>0.1</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>4-2</td>
<td>182.22</td>
<td>86.66</td>
<td>1.48</td>
<td>97.32</td>
<td>0.09</td>
<td>98.7</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>99.35</td>
<td>0</td>
<td>99.68</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>4-3</td>
<td>95.31</td>
<td>96.86</td>
<td>0.86</td>
<td>100</td>
<td>0.07</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>4-4</td>
<td>46.58</td>
<td>99.66</td>
<td>0.75</td>
<td>100</td>
<td>0.52</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>5-2</td>
<td>148.19</td>
<td>88.01</td>
<td>1.82</td>
<td>100</td>
<td>1.08</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>5-3</td>
<td>39.95</td>
<td>98.81</td>
<td>0.97</td>
<td>100</td>
<td>0.03</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>5-4</td>
<td>10.54</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6-2</td>
<td>146.91</td>
<td>92.35</td>
<td>1.92</td>
<td>100</td>
<td>0.01</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6-3</td>
<td>75.65</td>
<td>98.81</td>
<td>0.49</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>6-4</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 10: Reduction results varying querying types after applying one to six bloom filters
From Table 10, we found that all from our experiments, only after applying more than five bloom-filters, all the Ful_P and Ex_P are exactly the same as those of the reducer[32].

For each type of queries:

(2) For Ex_P, the type of lowest Ex_P is 3-2, which is 295.1% and the highest is 6-2, which is 0%.

(3) For Ful_P, the type of lowest Ful_P is 3-2, which is 42.79% and the highest is 6-4, which is 100%.

(4) In most case, both Ex_P and Ful_P are increased while the number of join attributes is increased with the same number of relations.

The best performance type is 6-4 because 99% of testing queries is Null Query.

6.3.3 Comparison of the results according to number of join attributes and number of bloom-filters.

(1) The results show that Ex_P decreased while increasing number of join attributes. The highest Ex_P happened with two join attributes and applying one bloom-filter which is 193.11%.

(2) The results show that Ful_P increased while increasing number of join attributes. The worst Ful_P is 77.45% which happen at one bloom-filter and two join attributes.
(3) For two join attributes, we need to apply at least two bloom-filters because there is huge difference in between, which are 193.11% and 2.64%.

<table>
<thead>
<tr>
<th>No. Of Hash Functions</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. Of Join Attributes</td>
<td>Ex_P</td>
<td>Ful_P</td>
<td>Ex_P</td>
<td>Ful_P</td>
<td>Ex_P</td>
<td>Ful_P</td>
</tr>
<tr>
<td>2</td>
<td>193.11</td>
<td>77.45</td>
<td>2.62</td>
<td>95</td>
<td>0.44</td>
<td>97.7</td>
</tr>
<tr>
<td>3</td>
<td>66.78</td>
<td>96.82</td>
<td>1.22</td>
<td>99.95</td>
<td>0.11</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>20.57</td>
<td>99.61</td>
<td>0.39</td>
<td>100</td>
<td>0.16</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 11: Reduction results of varying number of join attributes and number of bloom-filters.

(4) The largest average increment: for Ex_P it is 86.26%. And for Ful_P, it is 11.08% which can be calculated within one bloom filter column in Table 11.

6.3.4 Comparison of the results according to number relations and number of bloom filters.

(1) The results show that Ex_P decreased while we increase number of relations. The worst Ex_P happened with three relations and one bloom-filter, which is 135.47%. It is not ideal reduction. However, if we apply two bloom filters, Ex_P dramatically drops to 3.05% which is acceptable.
(2) The results show that Ful_P increased while increasing number of relations. The worst Ful_P is 78.12% which happens with one bloom-filter and three relations.

<table>
<thead>
<tr>
<th>No. Of Filters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. Of Relations</td>
<td>Ex_P</td>
<td>Ful_P</td>
<td>Ex_P</td>
<td>Ful_P</td>
<td>Ex_P</td>
<td>Ful_P</td>
</tr>
<tr>
<td>3</td>
<td>135.48</td>
<td>78.13</td>
<td>3.05</td>
<td>94.13</td>
<td>0.32</td>
<td>97.38</td>
</tr>
<tr>
<td>4</td>
<td>108.04</td>
<td>94.39</td>
<td>1.03</td>
<td>99.11</td>
<td>0.23</td>
<td>99.57</td>
</tr>
<tr>
<td>5</td>
<td>76.23</td>
<td>95.61</td>
<td>0.86</td>
<td>100</td>
<td>0.17</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>74.19</td>
<td>97.06</td>
<td>0.81</td>
<td>100</td>
<td>0.01</td>
<td>100</td>
</tr>
</tbody>
</table>

Table12: Reduction results of varying number of relations and number of bloom-filters.

(3) For three relations in a query, we recommend to apply at least two bloom filters because there is huge difference in between, which are 135.47% and 3.05%.

In the chapter, we analyze out experimental results regarding to those factors such as selectivity, connectivity. The finally chapter will provide conclusion and future work.
CHAPTER 7 CONCLUSION AND FUTURE RESEARCH

In the research, we propose a new reduction approach which uses multiple bloom-filters to process general queries. The approach aims to minimize both data transmission time and local processing cost by reduce relation sizes before performing JOIN. The evaluation of our algorithm is against the reducer [32]. We perform a mount of experiments to determine how the number of bloom-filters affects the performance of evaluation. Also the experiments would simulate general queries from varying selectivity, connectivity, number of relations and number of join attributes.

7.1 Conclusion

(1) To achieve exactly the same reduction effect as reducer [32], we need to apply at least six bloom-filters.

(2) In general, after applying one bloom-filter, the extra percentage against the reducer[32] is 95.51%. This is not ideal results because it need almost double size than those produced by the reducer[32]. However when we apply two bloom-filters, the Ex_P is only 1.41%. The difference of Ex_P between one bloom-filter to two is 94.11% which it is significant. The average percentage decreased on Ex_P from two to five bloom-filters is 0.47% which is slight. Therefore if utilization of more filters is expensive we can use two bloom-filters to achieve acceptable reduction performance.
Table 13: Final reduction results of applying one to six bloom-filters

(3) For the percentage of fully reduced queries, the lowest one is 83.38% which one bloom filter is applied. The highest Ful_P is 100% which we use six filters or more. The average increment of Ful_P is 3.16%(Table 13).

(4) In our experiments, 95% of random queries are null query. Above 93% of queries is fully reduced if a query is null query. Therefore, even though we simply apply one non-perfect hash function, we can detect 93% null queries before they are sent out. As a matter of fact, we can save a lot of local process time and transmission time. Especially for those queries with high connectivity, more relations and more joining attributes involved.

(5) From table 8, for all those number of bloom-filters which is less than six, we found that the extra percentage of reduction data size decreases and the percentage of fully reduced queries against that generated by the reducer [32] increases while their
connectivity increasing. We conclude that the higher connectivity, the better reduction performance. Further more, according to table 10, unlike connectivity, we found the higher the selectivity, the better reduction performance.

(6) The number of join attributes is also an important factor to affect the performance of our reducer. The experiments indicate that the more number of join attributes, the closer the final reduction data size to that of the reducer [32]. The worst extra percentage of reduction data size is 193.11% which is produced in two join attributes and one bloom-filter. The largest average decrement of Ex_P is 86.25% which is significant. This situation happened with two join attributes and one bloom-filter.

(7) For the number of relations, we found that for each query, the more number of relations, the closer performance against that of applying the reducer [32]. The largest Ex_P is 135.47% when we apply just one bloom-filter and three relations.

(8) In short, if we apply six or more bloom-filters, the results are exactly the same as that of reducer [32]. Otherwise, in order to obtain better reduction performance, we suggest to adopt those queries with higher connectivity, higher selectivity, more number of relations and more number of join attributes.
7.2 Future work

In the paper, we use multiple bloom-filters. Therefore, unavoidable, the collision is significant if applying just one bloom-filter. The future research could apply the reduction approach to relevant semijoin algorithm, like SDD-1[3] or AHY[1] etc., to reduce the data transmission time in Distributed Database Systems. Also, another research direction could be how to reduce collision with less bloom-filters. For example, we certainly can apply more than one hash functions to each bloom filter.
Appendix A: The techniques used in experiments

1. Related terms[31][32]:

1.1 Selectivity: The ratio of distinct attribute values over the attributes domain size.

1.2 Connectivity: the ratio of the number of join attributes in all relations over the product of the number of relations and the number of common joining attributes.

1.3 Cardinality: The number of tuples in a relation.

1.4 Domain: The total number of possible distinct attribute values.

1.5 “qspects”: A data file which contain the information of selectivity, connectivity, cardinality and domains.

1.6 “qscript”: A Tcl script program which can generate query

1.7 “build”: A C program which generates relations based on the statistics generated in qscript.

1.8 Per: A C program based on “Perfect” Algorithm.

1.9 opt: A C program based on our Algorithm. It contains eight different non-perfect hash functions.

1.10 RUNOPT: A shell script program which can do our experiments automatic as many times as we expect.
The execution diagram of program "RUNOPT"

1. Retrieve information
2. Generate one query
3. Generate relations
4. Run full reducer
5. Run Optimizer with "Perfect" Algorithm
6. Run Optimizer with our Algorithm.
   Repeat 8 times to apply one to eight non-perfect hash function

If repeat times == 8

True

False


[47] J.M. Morrissey, "File Structure", lecture notes for class 255, Computer Science Department, University of Windsor, Canada

[48] J.M. Morrissey, "Notes", Computer Science Department, University of Windsor, Canada


VITA AUCTORIS

Jianwei Wang was born in 1963 in ChangSha, HuNan, People's Republic of China. He graduated from Beijing ShiFan University Fu shu Middle School in 1983. From there he went on to the Beijing University, Beijing, P.R. China where he obtained a B.Sc. in Mechanics in 1987. He is currently a candidate for the Master's degree in Computer Science at University of Windsor and hopes to graduate in Summer 2000.