QUERY OPTIMIZATION HEURISTICS FOR CONTENT-BASED IMAGE DATABASE

BY

SEIFU GELETA FEYSSA

THESIS

SUBMITTED TO THE SCHOOL OF GRADUATE STUDIES OF ADDIS ABABA UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

June, 2004

ADDIS ABABA UNIVERSITY SCHOOL OF GRADUATE STUDIES

DEPARTMENT OF COMPUTER SCIENCE

QUERY OPTIMIZATION HEURISTICS FOR CONTENT-BASED IMAGE DATABASE

BY

SEIFU GELETA

NAME AND SIGNATURE OF MEMBERS OF THE EXAMINING BOARD

DR. SOLOMON ATNAFU, ADVISOR ____________________

____

____

In Memory of my mother and my father

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Dr. Solomon Atnafu for his advice and support in this thesis. His valuable previous work has served as a basis for this thesis.

I would like to acknowledge the Unity University College community, special thanks to members of Information Technology & Computation Science Faculty.

I would also like to thank my colleagues Dawit Bulcha and Fekade Getahun who are working their thesis on multimedia data management. The discussions I had with them have helped me learn more and improve quality of my thesis.

My deepest gratitude and sincere thanks goes to w/ro Amakletch Bogale and her family for their support through out my education.

Last but not least, I would like to thank my sister and the rest of the family for their support during my study.

Table of Contents

1. INTRODUCTION 1

1.1 BACKGROUND 1

1.2 PROBLEM STATEMENT 4

1.3 MOTIVATION 4

1.4 METHODS 6

2. RELATED WORK 9

2.1 OVERVIEW OF QUERY PROCESSING 9

2.2 QUERY OPTIMIZATION TECHNIQUES 9

2.2.1 Cost-based Optimization 10

2.2.2 Heuristic-based optimization 11

2.3 DIFFERENT QUERY OPTIMIZATION APPROACHES 12

2.3.1 System R optimizer 12

2.3.2 Starburst Optimizer 13

2.3.3 Volcano Optimizer 14

2.4 EXTENSIONS FOR OBJECT RELATIONAL-DBMS 16

2.5 QUERY PROCESSING IN CONTENT-BASED IMAGE DATABASE 18

2.5.1 Image Feature Extraction and Representation 18

2.5.2 Content-Based Image Database 19

2.5.3 Query Language for Content-Based Image Queries 20

2.5.4 Similarity-Based Query Processing 21

2.5.5 Indexing Structure for Image Contents 22

2.5.6 Query Optimization in Content-Based Image DBMS 23

3. OVERVIEW OF SIMILARITY-BASED ALGEBRA & COST MODEL 25

3.1 THE IMAGE DATA REPOSITORY MODEL 25

3.2 OVERVIEW OF SIMILARITY-BASED ALGEBRA 26

3.3 GENERAL TRANSFORMATION RULES ON SIMILARITY-BASED ALGEBRA 29

3.4 COST MODEL FOR CONTENT-BASED IMAGE QUERIES 30

4. QUERY OPTIMIZATION METHOD FOR CONTENT- BASED IMAGE DATABASE 36

4.1 GENERAL QUERY OPTIMIZATION FRAMEWORK 37

4.2 QUERY REWRITING HEURISTICS 39
4.3 INTEGRATING CONTENT-BASED IMAGE QUERY OPTIMIZER INTO IMAGE DBMS .. 51
4.4 MODELING A QUERY OPTIMIZER FOR CONTENT-BASED IMAGE DATABASE 54
4.4.1 Requirements 54
4.4.2 Design of Content-based image Query Optimizer 56
4.5 COMPONENTS OF THE QUERY OPTIMIZER 59
4.5.1 Search space 59
4.5.2 Search strategy 62
4.5.3 Cost Model 63

5. EXPERIMENTAL RESULTS 65

5.1 EXPERIMENTAL SETTING 65
5.2 EXPERIMENTAL TESTS 66

6. CONCLUSION AND FUTURE WORKS 75

6.1 CONCLUSION 75
5.1 FUTURE WORKS 76

ANNEX. A DESCRIPTION OF CLASS FOR THE QUERY OPTIMIZER 77

ANNEX. B COST FORMULA 79

REFERENCES 82

LIST OF TABLES

TABLE 1-1 ORIGINAL DATA ON FILM ANNUALLY WORLD WIDE ____ 6

TABLE 5-1 RESULT OF ORDERING SIMILARITY-BASED AND RELATIONAL SELECTION PREDICATES ____ 67

TABLE 3 RESULT OF RULE-1 WITH DIFFERENT SELECTIVITIES OF RELATIONAL SELECTION 68

TABLE 5-3 RESULT OF AN EXPERIMENT ON PUSHING DOWN SIMILARITY-BASED SELECTION ____ 70

TABLE 5-4 RESULT OF EXPERIMENT ON ELIMINATING REDUNDANT EVALUATION OF PREDICATES ____ 72

TABLE 5-5 RESULT OF INDEX SEARCH ON FEATURE VECTOR ____ 74

LIST OF FIGURES

FIGURE 2-1 HIERARCHICAL INDEX STRUCTURE FOR IMAGE CONTENT [36] 23
FIGURE 3-1 MANAGING IMAGE RELATED DATA IN TABLE [19] 26
FIGURE 4-1 GENERAL QUERY OPTIMIZATION ARCHITECTURE 38
FIGURE 4-2 QUERY TREE FOR ORDER OF EVALUATION OF SIMILARITY AND METADATA PREDICATES 41
FIGURE 4-3 QUERY TREE FOR PUSHING SELECTION PREDICATES TO BASE TABLE. 43
FIGURE 4-4 PUSHING SIMILARITY-BASED SELECTION IN SYMMETRIC SIMILARITY-BASED JOIN 44
FIGURE 4-5 AVOIDING REDUNDANT PREDICATE EVALUATION ON MULTI SIMILARITY-BASED JOIN 47
FIGURE 4-6 SINGLE EVALUATION OF PREDICATES ON BASE TABLES OF SYMMETRIC SIMILARITY-BASED JOIN 49
FIGURE 4-7 EXAMPLE OF UNNESTING NESTED SIMILARITY-BASED QUERY 50
FIGURE 4-8 GENERAL ARCHITECTURE OF AN IMAGE DBMS [48]. 52
FIGURE 4-9 GENERAL ARCHITECTURE SHOWING INTEGRATION OF QUERY OPTIMIZER WITH IMAGE DATABASE 53
FIGURE 4-10 OBJECT ORIENTED DESIGN OF THE CBIDBQUERY OPTIMIZER 58
FIGURE 4-11 SCENARIO OF QUERY OPTIMIZER 59
FIGURE 4-12 QUERY THREE FOR DIFFERENT WAYS IN WHICH THE EXPRESSION CAN BE EVALUATED 60
FIGURE 5-1 RESULT OF EXPERIMENT ON ORDERING SIMILARITY-BASED & RELATIONAL SELECTIONS 67
FIGURE 5-2 RESULT OF RULE-1 WITH DIFFERENT SELECTIVITIES OF RELATIONAL SELECTION 68
FIGURE 5-3 RESULT OF EXPERIMENT ON PUSHING SIMILARITY-BASED SELECTION BEFORE SIMILARITY-BASED JOIN 70
FIGURE 5-4 RESULT OF EXPERIMENT ON ELIMINATING REDUNDANT COMPUTATION OF SELECTION ON MULTI SIMILARITY-BASED JOIN. 73
FIGURE 5-5 RESULT OF USE INDEX ON IMAGE FEATURE TO ACCESS BASE TABLE. 74

Abstract

Query Optimization Heuristics for Content-Based Image Database

By Seifu Geleta
Advisor: Dr. Solomon Atnafu
June, 2004

The goal of query optimization is to select an efficient query evaluation strategy. Query optimization is a widely studied problem in the context of traditional database systems. As a result many query optimization techniques have been developed and integrated into the query processing module of these database systems. Due to the differences in query processing strategy followed by content-based image database systems, query optimization in these systems is different.

In recent years, the number of image used in different application areas is increasing at a high rate. As a result, many content-based image retrieval systems and content-based image database systems have been developed. These systems have become important to manage image queries based on low level features when the size of images to be managed is small. However, as the size of image collection to be managed increases, these systems could not be efficient. This is because the underlying database do not provide query optimization scheme for similarity-based image queries.

In this thesis we developed heuristic-based query optimization techniques that transform declarative query posed on content-based image database into more efficient form. To test the performance of the proposed optimization techniques we conducted experiments. The result of our experimental tests showed that our optimization techniques have brought a significant cost saving by reducing execution time of queries.

In addition to the query optimization techniques, we proposed a method of integrating these query optimization techniques with existing content-based image database systems. As the query optimizer of commercial database systems that support content-based image processing are not fully extensible, we proposed to develop and integrate a query optimizer into the Image query processing module previously proposed. This requires the development of query optimizer that implements the optimization techniques. Hence, we have identified the requirement of query optimizer for content-based image database and presented the design of an extensible rule-based query optimizer.

Keywords: Query optimization, image database, similarity-based query processing, Optimization for content-based image database.

Chapter 1

Introduction

1.1 Background

Query optimization is the process of transforming queries into more efficient execution plans. Query optimization have been used as an important tool for improving performance of query processing in relational, object-oriented, object relational and other database systems where query processing is done on very large collection of data. As a result many query optimization techniques that considers the data models and algebra used have been proposed and implemented for these database systems. These systems have highly benefited from the query optimization techniques developed for them. As a result queries which otherwise could take longer time are optimized and made to be executed in fractions of seconds.

Recently, content-based image database have become increasingly important in many application areas including medicine, crime prevention, fashion and graphic design, publishing and advertising, historical research etc. As a result many commercial and experimental content-based image DBMSs and Content-based Image Retrieval (CBIR) applications have been developed. DBMSs that support content-based image query processing provide support to manage images based on their low level contents. They provide data Types such as BLOB, and Bfile to store images and their contents, functions to extract and store image contents, methods to store and retrieve images in the database system environment. These systems have become important to properly store and manage large collection of images based on their contents.

A survey conducted in [22] shows that image is becoming an important form of data in many application areas. The size of image produced and used in different application areas is increasing from time to time at a high rate. Some proportion of many companies database consists of images. Considering these importance of images in many application areas commercial database venders (such as Oracle, IBM and Informix) have incorporated a module for managing images based on their contents in database system.

These DBMSs however, do not have a query optimization scheme that transform queries involving similarity-based operations into more efficient form. Queries are processed according to a default execution plan which may not be optimal. The lack of query optimization scheme and the increase in size of collection of images makes query evaluation in these systems inefficient and slow.

Considering this problem, S.Atnafu et.al. [1] have proposed similarity-based algebra that can be used to internally represent and process similarity-based image queries. This algebra has important properties that can be used for query optimization.

The main goal of this thesis to develop query optimization techniques that use properties of this similarity-based algebra to transform similarity-based queries into more efficient form and propose the method of integrating these query optimization techniques with existing content-based image database systems.

Definitions of Heuristics

Heuristic method of problem solving has been used in different fields of computer science. In Arteficial Intellegence it is the most widely used approach. In database query optimization heuristic method is becoming an important technique of problem solving. Heuristic based query optimization uses heuristic rules that rewrite a given expression into a better form.

Definitions given to the term heuristic in different letritectures are given below.

The Oxford Universal Dictionary definition of heuristic

As an adjective, heuristic (pronounced hyu-RIS-tik and from the Greek "heuriskein" meaning "to discover") pertains to the process of knowing by trying rather than by following some preestablished formula. (Heuristic can be contrasted with algorithmic*.) The term seems to have two usages:*1

1) Describing an approach to learning by trying without necessarily having an organized hypothesis or way of proving that the results proved or disproved the hypothesis. That is, "seat-of-the-pants" or "trial-by-error" learning.

2) Pertaining to the use of the general knowledge gained by experience, sometimes expressed as "using a rule-of-thumb." (However, heuristic knowledge can be applied to complex as well as simple everyday problems. Human chess players use a heuristic approach.)

Other Definitions

"A rule of thumb or guideline (as opposed to an invariant procedure). Heuristics may not always achieve the desired outcome, but they are extremely valuable to problem-solving processes." (Wilson) [2]

"An approach to systems or problem-solving using rules based on business practice, experience, or expert intuition rather than quantitative optimization." [3]

" In mathematical programming, this usually means a procedure that seeks a solution but does not guarantee it will find one. It is often used in contrast to an algorithm, so branch and bound would not be considered a heuristic in this sense. In AI, however, a heuristic is an algorithm (with some guarantees) that uses a

[1] *The Oxford Universal Dictionary*, Third Edition, (1955).

[2] citl.tamu.edu/citl-glossary-main.htm

[3] www.risnews.com/glossary.htm

heuristic function to estimate the "cost" of branching from a given node to a leaf of the search tree.” [4]

1.2 Problem Statement

The main objective of this thesis is to develop query optimization techniques for content-based image database and propose a method of integrating them with existing image DBMS.

Specific Problems includes:

- Develop and/or formalize heuristic-based query optimization rules that use properties of similarity-based algebra and its transformation rules.
- Test performances of the proposed optimization techniques.
- Propose method of integrating the proposed query optimization techniques with the existing content-based image DBMS.

1.3 Motivation

In traditional database systems query optimization has been used to efficiently and effectively process queries on a large volume of data. Like in traditional database systems queries in content-based image database are expressed using declarative language. As a result complex queries that may not be efficient may be generated by users or applications running on content-based image database.

Query processing in content-based image database is based on a different query evaluation strategy than relational one; query evaluation is based on similarity between images based

[4] *carbon.cudenver.edu/~hgreenbe/glossary/H.html*

on their feature vector. Access methods (indexing structures) used for similarity-based image query evaluation are different form that used for relational query processing. This implies that relational query optimizers do not optimizer similarity-based image queries using the same query optimization strategy they use for traditional data.

Processing of content-based image query is more complex and costly. Image data is information rich and voluminous. An estimate of storage requirement of a typical x-ray image as reported in [22] shows that images require large storage when compared with attribute data.

> *"A slightly smaller estimate of the amount of storage required for a standard chest x-ray comes from the University of Pittsburgh, Clinical Multimedia Lab, which estimates the size for a radiograph (2K *2K * 16 bits) as approximately 8 megabytes (uncompressed)"*

Storing images in database involves extracting their feature, transformation to feature vector and storing both the image and its feature vector. Hence, processing similarity-based image query requires more memory to store images and its feature vectors, and computation time for similarity matching. Accessing images and their feature from disk is also more expensive both in terms of I/O operation and computation time.

On the other hand, the number of images produced and used in different application areas is increasing at a higher rate. According to a survey conducted in [22].

> *"There are over 2700 photographs taken every second around the world, adding up to well over 80 billion new images a year taken on over 3 billion rolls of film"*

An estimate for the amount of image produced annually on films and storage requirements is given in the following table [22].

Table 1-1 Original Data on Film Annually world wide

	Units	Digital conversion	Total Pet bytes
Photography	82,000,000,000	5 MB per photo	410
Motion Pictures	4,000	4 GB per movie	0.016
X-Rays	2,160,000,000	8MB per radiograph	17.2
Total:			**427.216**

The expensive nature of similarity-based image query processing and the increase in size of images to be managed in the database requires effective and efficient query optimization techniques. Therefore, the aim of this thesis is to develop heuristic based query optimization techniques that transform queries posed on content-based image database using declarative language into an efficient form.

1.4 METHODS

To undertake this research work, the following techniques will be used.

a) **Review literature**

- Related works in the areas of relational, and object relational database systems will be explored in order to understand the techniques that are adopted to optimize queries in these systems.
- Related works in content-based image database and similarity-based query optimization will be studied.
- Related works in building query optimizers will be studied.

- Different methods, techniques and standards pertaining to similarity-based image retrieval will also be explored in more details.

b) Develop and/or formalize Query optimization

Query optimization heuristics that transform Similarity-based queries into more efficient form will be developed.

c) Performance evaluation

The performance of proposed algorithms and formalisms will be tested by implementing them on prototype system in an appropriate environment.

d) Propose method of integrating query optimization techniques with existing database.

A method for integrating the proposed query optimization techniques with existing content-based image database will be explored and proposal of how to integrate them will be made.

Organization of the thesis

Chapter two review related works on query optimization in relational, and object relational database system. It also presents related works on query processing and optimization in content-based image database. In chapter three we will present an overview of the image data repository model, and similarity-based algebra on which this work is based. We will also present the cost model and selectivity estimations for the algebraic operators used in this work. Chapter four will present our query optimization methodology, the architecture of proposed query optimizer for content-based image database and the query optimization techniques used. In chapter five we will present the experimental result of proposed query optimization heuristics and the last chapter will present the conclusion and the future works.

Chapter 2

Related Work

2.1 Overview Of Query Processing

Query processing is the process of answering queries posed by users or user application. Query processing is done by the component of DBMS known as query processor. Query processor consists of two important components; the query optimizer and the query execution engine. The query optimizer takes parsed representation of SQL query as an input and is responsible for generating an efficient execution plan that correctly and efficiently answers the input query. Since the number of equivalent algebraic representations of a given SQL query can be very large depending on the properties of the logical algebra and there may be many implementations algorithms for each of the logical algebra operators, the task of query optimizers is complex problem of searching an optimal plan from large search space.

The query execution engine takes an optimal query execution plan selected by the query optimizer and executes it to answer the query. Query execution plan is represented in an operator tree where each nodes of an operator tree represents an algorithm that efficiently implements a given logical operator.

2.2 Query Optimization Techniques

Query optimization involves selection of an optimal query execution plan from among a large number of equivalent plans. As the number of equivalent plans can be very large, the problem of selecting an optimal plan is NP-hard. As a result, there is always a tradeoff between extending the search to all search space and an overhead of searching an optimal plan. Extending a search to the entire search space produces an optimal plan but also results in a higher runtime overhead. This makes the optimizer miss its goal of reducing execution time as it spends much time searching for an optimal plan rather than executing the query. Hence, most query optimization approaches restrict the search space to a certain subspace where an optimal plan is supposed to exist. System R optimizer, for example, restricts the search space to a left deep tree subspace. The plan obtained from this optimization approach is not optimal but more efficient than those plans that fall in the domain of considered subspace [24].

Query optimization is performed by employing a certain technique that tries to select optimal plan. The two most widely used optimization methods are heuristic-based and cost-based optimization.

Heuristic-based optimization uses properties of the underlying logical algebra to transform queries into more efficient form. They make certain transformations to the logical query plan to reduce the size of the result before computationally complex operations.

Cost based optimization on the other hand estimates the cost for each possible physical query plans using a cost model designed for the underlying database system. Some approaches combine these two techniques by reducing the potential plans by heuristic rules and using costs in making the final decision about the query plan [24].

2.2.1 Cost-based Optimization

In cost-based optimization method, the cost of each operator in the expression are calculated and combined to give the overall cost of the plan. The costs of different execution plans are estimated with a specific cost function, and the plan with the lowest cost estimate is chosen as the final plan. Since the cost function is an estimate, the resulting plan is not necessarily always an optimal solution [24].

2.2.2 Heuristic-based optimization

Heuristic-based query optimization operates on the logical query plans which consist of a sequence of algebraic operators. Algebraic operators are usually represented as a query tree or query graph. The query optimizer takes an initial logical plan from the parser and uses heuristic rules to transform them into an equivalent form that can be processed more efficiently.

These heuristic rules try to minimize the size of the intermediate results and the number of rows that are scanned during the query processing.

In relational database query optimizers, some of the important heuristic rules include [37]:

- Perform operations that restrict the result before the operations of high complexity. For example, selections operations are pushed down to restrict the size of data on which relational join is applied.
- Combine a sequence of projections and selections into a single projection or selection, respectively. These combinations may avoid repetitive accessing of the same table and provide access paths that are not available with a single operation.
- Break up a conjunctive selection into a sequence of selections. This rule gives more freedom to the movement of operations in a query tree.
- Combine each Cartesian product with a subsequent selection into a join if possible.

In real world query optimization, heuristic rules alone do not always result in optimal query plan. Heuristic rules operate only on logical algebra expressions. Hence most optimizers use both heuristic and cost based optimization techniques.

2.3 Different Query Optimization approaches

In the context of relational database systems different query optimization approaches have been proposed. The representative ones are System-R, Starburst, and Volcano optimizers.

2.3.1 System R optimizer

One of the earliest query optimizers proposed and implemented is the System R query optimizer. System R optimizer is the first to show the practical applicability of query optimization in a commercial environment. Most of the current commercial Relational Database Management Systems use the query optimization philosophy proposed for system R.

Important features of System R optimizer includes:

- Optimization technique employed is cost-based. The cost model is used to assigns an estimated cost to any partial or complete plan in a search space. The cost model is based on; statistics maintained on relations and indices, formula to estimate the CPU and I/O costs, and formula to estimate selectivity of predicates [23].

- The enumeration algorithm is based on dynamic programming and the idea of interesting orders. The idea of interesting orders was used to generate only those access plans in the search space that were likely to be part of other access plans.

- The search strategy is restricted to consider only left deep operator trees (in which the inner relation was always a base relations), and by delaying cross products as far as possible [23].

- The search strategy employed was a bottom up exhaustive search strategy with dynamic programming. In a bottom up strategy children of nodes in an operator tree are optimized before the node itself.

- The join algorithm can use either Nested loop or sort merge implementation.
- The base relations can be scanned either using index scan or sequential scan.

2.3.2 Starburst Optimizer

Starburst is a rule-based extensible query optimizer developed at IBM Alemaden research center. Query optimization in starburst follows two stages; query rewrite and plan optimization.

The query rewrite phase of optimization operates on the Query Graph Model (QGM) which is introduced for internal representation of algebraic expressions [27].

Query rewrite phase uses heuristic rules to transform a given logical QGM into an improved form.

Rules in starburst query optimizer have two components:

- *Condition*: The condition checks the applicability of the rule.
- *Enforcer*: Transforms the QGM into equivalent QGM if the rule is applicable [27].

Some of the important query rewriting techniques (heuristic rules) employed includes:

- *View merge*: This technique involves merging of two or more views into one. It makes additional join orders possible and can eliminate redundant joins.
- *Subquery to join transformation* (Unnesting of nested queries): This technique is employed to removes restrictions on join method/order and improves efficiency. In nested queries, for each record of the outer selection, the inner selection is performed once. It is not convenient to select join order or join method.

- *Predicate Pushdown:* this technique uses the commutative property of the logical algebra to order operators. With this technique, more selective and cheaper predicates are evaluated early on base relations.

- *Distinct Pullup and Distinct pushdown*: This technique allows optimizer to eliminate duplicates computations.

The second phase of optimization is plan optimization. During this phase of query optimization the execution plan for QGM is chosen. The physical operators are combined in a variety of ways to implement the higher level operators.

The search strategy employed by starburst is a bottom up dynamic programming algorithm.

The starburst query optimizer is extensible in a sense that new algebra and new rules can be added into the optimization module.

2.3.3 Volcano Optimizer

Volcano is a rule based query optimizer generator that is designed to be flexible and extensible to specific database architectures. Volcano optimizer generator uses two kinds of rules;

- *Transformation rules*: Transformation rules transform or map a given algebraic expression into an equivalent but more efficient algebraic expression.

- *Implementation rules:* map an algebraic expression into an operator tree (implementation algorithm).

The enumeration algorithm employed by volcano is a top down query dynamic programming algorithm. In a top down strategy parents of nodes in an operator tree are optimized before the children nodes [29].

The search strategy is exhaustive, meaning that all operator trees that are generated by application of rules are evaluated before an optimal plan is returned. Dynamic programming is used to prune the search space as much as possible. The process of pruning is similar to the one in System R.

Query optimization techniques discussed in this section are those proposed for relational database. These techniques do not directly apply for content-based image database query optimization. This is because of the differences in query processing strategy followed by content-based image database.

Content-based image query processing is different form relational query processing in many respects. Some of these are:

- Unlike relational query processing which is based on exact match, query processing in content-based image database is based on the degree of similarity between images based on their feature vector.

- The cost model for similarity-based image database is different form the relational one.

- Access methods designed for relational database are not applicable for similarity-based image query (B-tree index is not applicable for indexing image contents).

- Relational algebra is designed for attribute based data and can not be used for similarity-based query processing. Similarity-based queries are internally represented in similarity-based algebra [1].

Therefore, query optimization for content-based image database requires optimization techniques that consider these factors. Similarity-based algebra and review of cost model for content-based image database are given in chapter3.

2.4 Extensions for Object Relational-DBMS

Object Relational DBMS aims at achieving the functionality of object data management in their unique way. They try to support object-oriented capabilities to the relational DBMS technology while at the same time preserving full relational capabilities [21].

Object relational query optimizers use the same query optimization techniques defined for relational query optimizers but extended to handle added object oriented issues. Stonebreaker [29] has identified 16 specific extensions that must be made to traditional optimizer so that it will work well in an object relational environment. Some of the important object relational optimizer's extensions are discussed below:

1. Indexes on complex, user defined types: ORDBMS must support building of indices on user defined types. With the added index the query optimizer can choose to use the introduced index instead of sequential scan to improve query evaluation [29].

2. Costing user defined operators: For each user defined types, ORDBMS query optimizer should be told to use appropriate operators defined for them. This includes, user defined comparator, user defined negatrons etc [29].

3. User defined selectivity functions: In relational DBMS selectivity functions are hard-coded into the optimizer. When new types and their comparison operators are introduced, the optimizer has no possible knowledge on the selectivity of the operators for the new data type. ORDBMS optimizers should be extensible to allow one to specify how the optimizer computes selectivity [29].

4. Optimization of expensive functions: In relational optimizers, the restriction will be always performed first on base relations. However, when expensive clauses appear in a query which is common in ORDBMS, this strategy may be undesirable. Hence the query optimizer should be smart enough to apply restrictions only when it is helpful. Techniques such as predicate migration algorithms may be used for this purpose [29].

5. Operator and function notations: Operators and functions having the same semantic meaning may be represented differently. For example, the operator “>” may also be represented using the operator “greaterthan” in another query expression. Even though, these two representations have different synthetic forms they are the same. An object relational optimizer should be able to deal with such conditions [29].

6. B-Tree and user defined comparison: B-tree can support index scan for the operators {<, <=, =, >=}. For user defined types one must be able to specify the definition of these operators. With generic B-tree operator classes, it is possible to build a B-tree index that supports index scans on a wide variety of user defined data types and operators [29].

7. Access Method on a function of a data: Traditional relational systems the B-tree index is built on the values of the attribute. Object relational database systems provide complex data types such as images where it is not relevant to build an index on the actual attribute. “For example, a B-tree index on an image data type keeps the index in a sorted order on the bytes in the image. This is not relevant to any user queries. Thus the only plausible indexes are on function of the image data” [29].

8. Smart Ordering of the clauses in a predicate: Traditional relational optimizer evaluates clauses in a predicates from left to right. This strategy is reasonable in traditional SQL queries as all clauses are simple and not much expensive. In object relational systems some predicates may be much more expensive than the others in a give query expression. Hence the optimizer should have more information on resource utilization of predicates so that it determines in which order to evaluate them [29].

9. User-Defined Access Methods: B-tree indexes are one dimensional access methods and are not useful for data that is stored multidimensional space. It must be possible to add new access methods such as multidimensional indexing for applications that requires it [29].

2.5 Query Processing in Content-Based Image Database

In this section we will discuss query processing and optimization issues in content-based image database.

2.5.1 Image Feature Extraction and Representation

Traditionally, image retrieval was based on annotation of the images using keywords. In this method the image is first annotated with keywords (textual descriptions that identify the image e.g. date the image was taken, the domain to which it belongs, photographer, devices used) and stored in tables of database management systems. Retrieval of the image is managed using traditional DBMS using index on the keywords. This method, however, couldn't be effective as the size of image data to be managed increases for the following two reasons. First, as the size of the image becomes large, manual annotation requires large labor. Second, the perception of the content of the image is subjective that means the same image is perceived differently by different people and this will make managing image data very difficult. As a result, image retrieval technique that is based on automatic extraction and storage the content of the image was proposed as an alternative or to be used in combination with annotation method.

Visual feature based image retrieval is based on the visual features (low level content) of the image, such as color, texture, and shape. In this technique, first, when the image is loaded into the database the low-level features of the image are extracted using feature extraction method and then their feature vectors are stored in the database. The feature vector represents the mathematical representation of the color, texture, of the feature of the image.

During content-based retrieval of images, the features of the query image are extracted and compared with the feature of each images stored in the image table. Those images in the database that satisfy the similarity condition are returned as a query result. For matching of similarity between images based on their feature vector, different techniques have been proposed. The most popular are Euclidean metric and maximum metric algorithms.

2.5.2 Content-Based Image Database

In content-based image database, image is represented by its features of color, texture, shape etc. in the form of feature vectors. The feature vectors of images require complex data structure for their representation. Due to the requirement of complex data structure for extracted image features and user defined methods for manipulating their properties, relational database management systems provide very limited support to manage images based on their low-level features [9].

Commercial object relational database management systems such as oracle9i, Informix and DB2, have started providing capability for defining complex user defined types (UDTs) and user defined functions (UDFs) for representing the structure of image features in a database and manipulating on properties of images [9]. These systems do also provide mechanisms for calling user defined functions directly using SQL statements. Additionally, Object relational database management systems provide built-in data types such as BLOB and CLOB for storage of large multimedia objects.

Oracle 9i, DB2 and Informix have integrated image retrieval module; oracle interMedia, DB2 image extender and Informix data blade respectively. In these systems, image processing operations are performed using special modules.

An image data repository model proposed in [2] allows the features of the image and its text attributes to be stored with the image in image table. The data model allows querying images based on image content, text attribute and combination of both. The similarity-based algebra proposed in this paper has a very important role for facilitating content-based processing of similarity-based image queries. The prototype of this data repository model is implemented on Oracle9i. In this thesis we will consider this similarity-based algebra as a basis for our proposal on query optimization.

2.5.3 Query Language for Content-Based Image Queries

Query processing in content-based image database is based on similarity between images on their features. As a result SQL languages (SQL2) which is designed for relational database do not provide full facilities to specify similarity-based query. With this consideration many query languages such as MOQL [20], FOQL [5] have been proposed in literatures. These languages are however developed for a specific domain and do not apply for all areas.

The current version of SQL that is SQL3 has added features that can be used to manage images based on their contents. These new features include:

- The ability to store large objects including images and other multimedia and large document types.
- Ability to define nested columns i.e. Group of columns within a column.
- Ability to define complex data types that can represent contents of image data.
- Complete library or set of data structures and routines that support still images and types and operations on these data types.

These added features enables images to be managed in database system based on their content using declarative languages (SQL).

2.5.4 Similarity-Based Query Processing

Querying traditional database is different form querying a content-based image database. In traditional document database records that exactly match to the query criteria are returned. In content-based image database however, instead of exact match, all images that are similar to the query image with a specified degree of similarity are returned as a query result.

Similarity-based query processing is based on similarity of images on their features. When an image is inserted into database its content is extracted using feature extraction functions and stored with the image in database as an additional attribute of the images. The features of the image may be indexed by using a multi dimensional indexing methods to facilitate content-based query processing. Similarity-based query processing is then based on the image features not on the actual image.

A query posed on content-based image database among other things contains the query image, value of features (feature coefficients) and the threshold values. During similarity-based query processing contents of the query image is extracted and then matched with the feature of images in the image table. Those images stored in the image table that match the query image with a given threshold will be retrieved as the result of the query.

As it was discussed above, query processing on content-based image database is not exact match rather it is based on degree of similarity between images. Hence the query model of traditional database systems is not directly applicable. To commonly used query models for content-based image database are range query model and k-nearest neighbor (k-NN) query models.

The k-NN query model returns k top images that are more similar to the query image than any other images in the image table. In this method processing of query of the form "retrieve images similar to a query image q", involves extracting the features of q, matching it with feature of all images in the database and retrieving k top images that are similar to q in sorted order. The k-NN model always returns k images even though they are not similar. Hence the user can always get k images in response to his query even though they are not similar.

Range query model on the other hand returns all images in the image table that falls in a given threshold. The threshold value is given by the user and it shows the maximum distance between the value of feature of the query image and the result images. Range query may return no image at all or all images depending on the value of the threshold and the distribution of the image collection.

In addition to these simple similarity-based query processing functions, operators for complex similarity-based image queries involving more than one image tables have been proposed in [1]. These operators includes; Similarity-based selection, similarity-based join, etc. since our work is based on this similarity-based algebra we will give detailed description in chapter 3.

2.5.5 Indexing Structure for Image Contents

An indexing structure facilitates searching of data by limiting the accessed data size and the comparisons made. It is an important component of modern database systems for processing efficiency. As it is discussed above, in content-based image query processing when the image is stored in the database its important features are extracted and mapped into high dimensional feature vectors. Content-based query for the query image is then done by performing range or k-NN search in a high dimensional feature space. for image databases that consists large number of images it is essential to use appropriate multidimensional indexing technique to achieve efficient searching of images. Structurally most of high dimensional indexing methods are based on hierarchical clustering of the data space. There is a single root node that serves as an entry point for query processing and update. The data vectors are stored in the data node such that spatially adjacent vectors are likely to reside in the same node. Data nodes are stored in a hierarchically structured directory. Each directory node points to set of sub trees. Directory nodes consist of (key, pointer) tuples, in R-tree, its variants and X-Tree the key are bounding rectangles. The trees are height balanced i.e. the length of the path from any data page and the root is identical. This height is called height of the index structure.

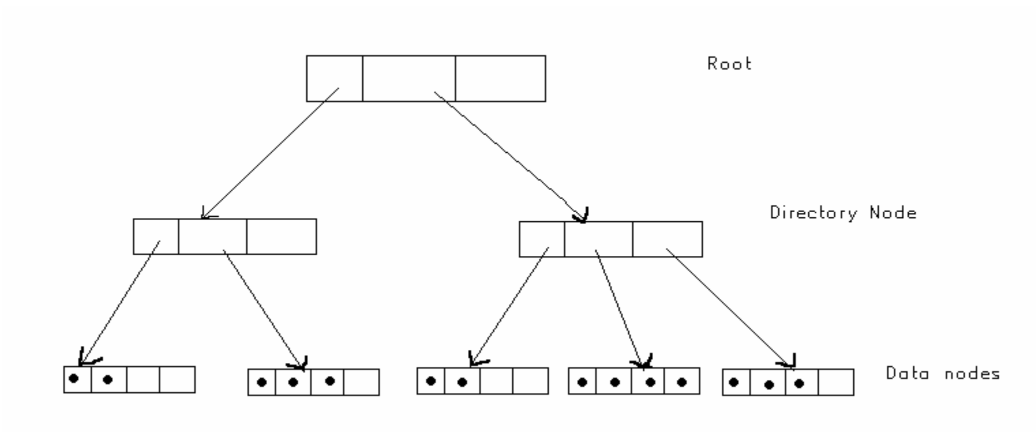


Figure 2-1 Hierarchical index structure for image content [36]

2.5.6 Query Optimization in Content-Based Image DBMS

Many commercial object relational database systems such as Oracle, DB2 and Informix have included a module for managing images based on their low level contents. These systems have provided a facility for storing images and their contents and for querying images based on their contents.

However, these systems do not have query optimization module for content-based image queries that transform queries into efficient execution plans. Queries are evaluated by the default expression. The default expression processed may require more cost than it worth (may not be an optimal one). As a result query processing in this domain is not efficient and the query response is slow.

The first approach for query processing in content-based image database based on algebra is the one described in [1]. This work have introduced data repository model and similarity-based algebra for content-based image database. It also discussed important properties for the proposed similarity-based algebra that are useful for query optimization. The optimization of symmetric similarity-based join using the *mine* operator is also described in [1].

What is currently missing from content-based image database is a technique for query optimization. The goal of this work is develop query optimization techniques for content-based image database. The query optimization approach we follow is an algebraic optimization of declarative queries containing similarity-based operations. Hence, we will develop heuristic rules that transform similarity-based queries on content-based image database into a more efficient form. We will propose a method for integrating these query optimization techniques with existing image database systems.

Chapter 3

Overview of Similarity-Based Algebra & Cost Model

This thesis is based on the image data repository model and similarity-based algebra proposed in [1]. In this chapter we will give an overview of the image data repository model, and similarity-based algebra operators and a possible cost model.

3.1 The Image Data Repository Model

The image data repository model proposed by S.Atnafu et.al. [1] represents an image as a schema of five components M(id, O, F, A, P) that works under an object relational model.

ID is a unique identifier of an instance of M

O is reference of the image object

F is feature vector representation of object O

A is an attribute component that may be used to describe the object O using textual data or key words.

P is a data structure that is used to capture pointer link to instance of other image tables as a result of a binary operation.

The following figure shows how different image components are mapped to each components of the image table.

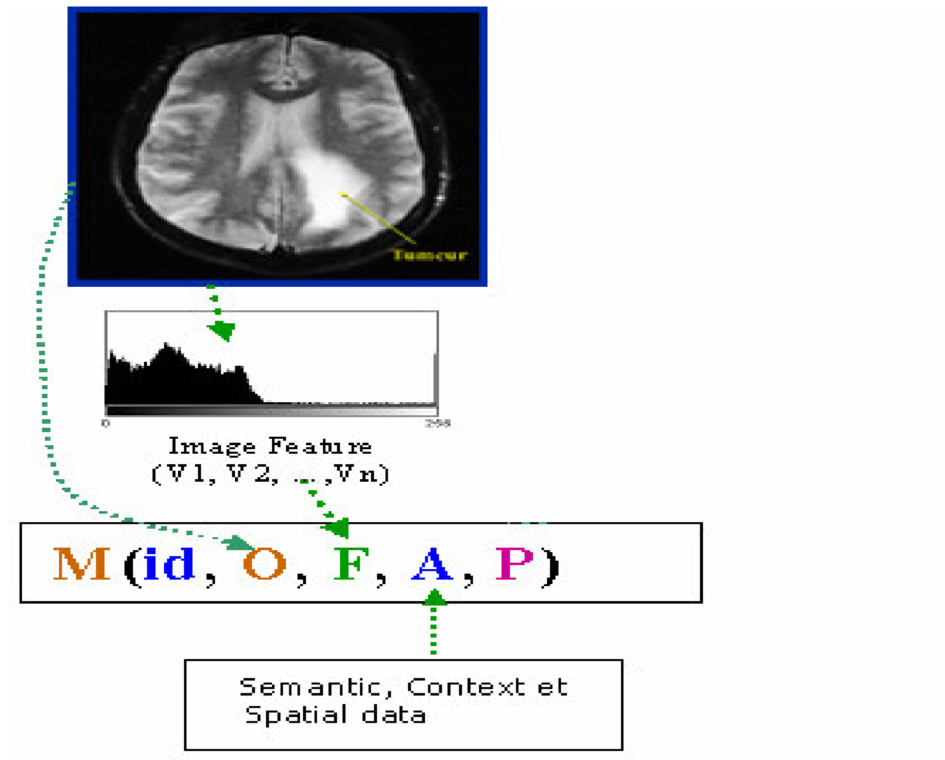


Figure 3-1 Managing image related data in table [19]

3.2 Overview of Similarity-Based Algebra

Algebraic operators

The similarity-based algebra, originally described in [1], is designed for managing image queries based on their low level contents. Some of similarity-based algebra operators and their properties are presented below.

Similarity-Based selection operator

$\sigma^{\varepsilon}_{x}(M)=\{(id,o,f,a,p) \in M \,/\, o \in R^{\varepsilon}(M,x)\}$

Where $R^{\varepsilon}(M, x)$ denotes the range query with respect to ε for the query image x and the set of images in the image table M.

Similarity-Based selection operator defined above is a unary operator that iterate on the feature vector component of the image table to select those images of a table that are similarly to the query image with a given threshold. The similarity-Based selection operator returns images instances of the image table where the image component that fall in the given range ε from the query image q.

Similarity-Based Join Operator

Given two image tables M1 and M2, the similarity-based join is formally given as [1].

$M_1 \otimes^{\varepsilon} M_2 = \{(id_1,o_1,f_1,a_1,p_1') \,/\, (id_1,o_1,f_1,a_1,p_1) \in M_1$ **and** $p'_1 = p_1 \cup (M_2, , \{(id_2, \|o_1 - o_2\|)\})$ **and** $p'_1 \neq Null\}$,

The similarity-based join operator denoted by $\otimes^{\varepsilon}$ is a binary operator that joins two images tables on their feature vector component. The resulting joined table contains images from the left table that have related image (similar to it with a give threshold) in the right table with their p component modified to contain reference to similar images in the right table. It is assumed that the feature vectors (F_v) component of the two input image tables have the same structure.

Properties:

- The *P* component of each row of the result table of the similarity-Based join operator contains reference for related image in the right table.

- If an image in the left table has no image similar to it in the right table i.e. its p component is null then the image will not appear in the resulting joined table.
- The order of input tables of the similarity-Based join operation is important i.e similarity-Based join operator is not commutative.

Additive Union

The additive union operator is used to merge two image tables. Given two image tables $M_1(id_1, O_1, F_1, A_1, P_1)$ and $M_2(id_2, O_2, F_2, A_2, P_2)$ .The Additive Union of M_1 and M_2 is defined as[1]:

$$M1 \overset{+}{\bigcup} M2 = \{(id, o, f, a, p) \mid (id, o, f, a, p) \in M1 \vee (id, o, f, a, p) \in M2\}$$

Symmetric similarity-based join

Symmetric similarity-based join on two image tables M_1 and M_2 denoted by $M_1 \oplus^{\varepsilon} M_2$ is the result of two one way similarity-based joins merged by the additive union joins. Symmetric similarity-based join is commutative.

$$M_1 \oplus^{\varepsilon} M_2 = (M_1 \otimes^{\varepsilon} M_2) \overset{+}{\bigcup} (M_2 \otimes^{\varepsilon} M_1)$$

***Mine* Operator**

As discussed above the symmetric similarity-based join is implemented by two similarity-based join operators. The *Mine* operator enables us to find the result of one similarity-based join after computing the result of the other. That is, Once $M_1 \otimes^{\varepsilon} M_2$ is done; $M_1 \oplus^{\varepsilon} M_2$ can be found by applying *Mine* operator. The *Mine* operator uses the P component of $M_1 \otimes^{\varepsilon} M_2$. Considering the similarity-based join $M_1 \otimes^{\varepsilon} M_2$ we can have the following properties.

$Mine(M_1 \otimes^{\varepsilon} M_2) = M_2 \otimes^{\varepsilon} M_1$

$Mine(M_2 \otimes^{\varepsilon} M_1) = M1 \otimes^{\varepsilon} M_2$

3.3 General Transformation Rules on Similarity-Based Algebra

Similarity-based algebra has important properties that can be used for rewriting of queries into more efficient form. The query rewriting rules we are going to develop in this work uses these properties to guarantee correctness of optimized logical expressions.

1 Pushing a similarity-Based selection into additive union

$$\delta^{\varepsilon}_{q}\ (M1U\ M2)\ \equiv\ \delta^{\varepsilon}_{q}\ (M1)\ U\ \delta^{\varepsilon}_{q}\ (M2);$$

2 Pushing similarity-Based selection into a similarity-Based join

$$\delta^{\varepsilon}_{q}(M_1 \otimes^{\varepsilon} M_2) \equiv \delta^{\varepsilon}_{q}\ (M_1) \otimes^{\varepsilon} M_2;$$

3 Pushing similarity-Based selection into symmetric similarity-Based join

$$\delta^{\varepsilon}_{q}\ (M_1 \oplus^{\varepsilon} M_2) \equiv \delta^{\varepsilon}_{q}\ (M_1) \otimes^{\varepsilon} M_2 \overset{+}{\cup} \delta^{\varepsilon}_{q}\ (M_2) \otimes^{\varepsilon} M_1$$

4 Pushing relational selection into an additive union

$$\delta_a(M_1 \overset{+}{\cup} M_2) \equiv \delta_a\ (M_1) \overset{+}{\cup} \delta_a\ (M_2);$$

5 Pushing relational selection into similarity-Based join

$$\delta_a\ (M_1 \otimes^{\varepsilon} M_2) \equiv \delta_a\ (M_1) \otimes^{\varepsilon} \delta_a\ (M_2)$$

6 Pushing relational selection into symmetric similarity-based join

$$\delta_a\ (M_1 \oplus^{\varepsilon} M_2) \equiv \delta_a\ (M_2) \oplus^{\varepsilon} \delta_a\ (M_1)$$

7 Commutatively of symmetric similarity-Based join

$$M_1 \oplus^{\varepsilon} M_2 \equiv M_2 \oplus^{\varepsilon} M_1$$

8 Exchanging order of similarity-Based join using the *Mine* operator

$$M_1 \otimes^{\varepsilon} M_2 \equiv \mathrm{Mine}(M_2 \otimes^{\varepsilon} M_1)$$

3.4 Cost Model for content-based image queries

Cost model is an important component of query optimizers. Query optimization techniques require knowledge of the cost of operators and selectivity of predicates for each operator. Cost-based query optimization uses the cost of operation of each operator in the query expression to select optimal query evaluation plan from among large number of equivalent expressions.

The ability to estimate the cost of query accurately and efficiently is of fundamental importance for cost based query optimizers. The cost and selectivity estimations should be accurate because they are used by the query optimizer to make decision for the selection of query execution plan.

Cost estimation involves collection of statistical information that has been stored in the database, estimation of intermediate result produced by each operators and estimation of cost of executing each operator in the query expression [39].

The cost of executing a query includes:

- Access cost to secondary storage,
- Storage cost to store intermediate results,
- Computation (CPU) cost, Memory usage cost, and
- Communication cost [37].

The communication cost applies only for distributed query processing. Cost estimation in the context of RDBMS requires the following specific database and system information;

- The number of records (tuples) in the data stream on the base table. This information is used to determine the cost of data scans, joins and their memory requirement.
- Size of disk pages in bytes.
- I/O cost of reading pages sequentially.
- The number of physical pages used by the table.

Some of these factors are system parameters (Hardware, operating system, DBMS configuration) and they are the same for all queries that are executed. Others are determined for each algebraic operator individually.

Estimation of cost of executing operators and selectivity of predicates on similarity-based queries in content-based image database requires more information than the traditional ones. In addition to the above stated cost factors, the cost of query execution in content-based image database depends on dimensionality of the data space, the distribution of the data and the value of the radius of the range query (ε).

The following sections will present the cost estimations for each of the similarity-based operators.

Cost Estimate of Similarity-Based Selection operator

Similarity-based selection operator returns images that falls in a given range from the query image. It applies a range query on all objects of the table to select those that are similar to the query image with a specified threshold.

Similarity-based selection can be implemented using index if it exists on the feature vector component of the image table or using sequential scan method. If there is no indexing structure on the feature vector component of the image table the Similarity-based operation can be implemented by a single sequential scan of the image table. In sequential scan implementation of similarity-based queries the data in the table is read in blocks which are then extracted to get the data and perform similarity matching. In this access method the cost of performing the operation is proportional to the size of the table in bytes [33].

The cost of performing similarity selection for sequential scan is proportional to the seize of data in the image table and it can be approximated as [33].

$$Sizeof(M)=d.N.x$$

Where M is an image table, d is dimension of the feature vector and N is the cardinality the table and x is size of memory (float).

To reduce the number of data accessed, the memory usage, and to facilitate similarity-based image query processing, multidimensional indexing techniques may be used. Multidimensional indexing structures reduce the amount of data accessed to perform similarity-based matching and hence reduce the cost of query evaluation. The cost formulae for similarity-based selection are given in Annex B.

Cost of similarity-Based join operation

Similarity-based join is one of the most important operators in content-based image database. It joins two image tables based on feature vector component of the image table and produces the result table that contains objects from the two tables that satisfy the similarity predicate in the given range.

With the absence of index on the feature vector of tables to be joined the similarity-based join can be implemented by sequential scan of the base tables. There have been proposals for constructing index on fly by repeated insertion of values into the index structure. However, this method of constructing index performs poorly and can not amortize by the performance gain during join processing. Another proposal for the construction on index on the feature of images is given by Bohm and et.al. [34]. They Proposed a dynamic bulk loading multidimensional index structure that has smaller runtime compared to run time of repeated insertion operation.

Cost of Multi similarity-Based join.

Multi similarity-based join operator joins more than two image tables based on the similarity predicates. It is implemented by pair wise similarity-based joins. Each pair wise join is combined with the additive union operator to make up the multi similarity-based join.

Multi Similarity-based join is defined as:

$$\mathbf{M_1 \otimes^{\varepsilon} M_2 \otimes^{\varepsilon} \ldots . \otimes^{\varepsilon} M_n = M_1 \otimes^{\varepsilon} M_2 \; U \; M_1 \otimes^{\varepsilon} M_3 \; U \; \ldots \; U \; M_1 \otimes^{\varepsilon} M_n}$$

The cost of a multi similarity-based join can be obtained by adding the cost of the pair wise similarity-based joins and the cost of performing the writing and reading intermediate results of each joins into disks for merging of these intermediate results if their size is more than available memory space. If the intermediate results are not written back to disk the cost of multi similarity-based join may be estimated as:

$$\mathbf{A}_{total} = \sum_{i=1}^{n} A_i + X_i$$

Where A_i the cost of performing the ith join and X_i is is the cost of performing the i^{th} additive union.

Cost of Symmetric Similarity-Based Join

Symmetric similarity-based join of two tables M_1 and M_2 given by the expression: $M_1 \oplus M_2 = (M_1 \otimes M_2) \overset{+}{\bigcup} (M_2 \otimes M_1)$ is the additive union of two similarity-based joins [1]. The cost of symmetric multi similarity-based join is the cost of the sum of the two one way similarity-based join operations and the cost of additive union operator.

Cost of symmetric Multi similarity-Based Join

The cost of performing symmetric multi Similarity-based join is obtained by adding the cost of individual pair wise symmetric similarity-based join operations and the cost of performing the additive union operations. If the intermediate result is written back to disk the cost includes the cost of writing to and reading from disk.

Cost of Entire Processing Tree

The cost of the entire processing tree is obtained by adding the cost of each nodes of the processing tree. The processing tree consists of either similarity-based selection operations and similarity-based join operations all joined by the additive union operations or symmetric similarity-based join, associated selection operations and additive union operation to join the result of each join operations. The later case may also involve the *Mine* operator to minimize the cost of query evaluation. In all cases the cost of evaluating the query is the sum of costs of evaluating each operator in the query expression.

Selectivity Estimation

In addition to estimating the execution cost of similarity-based queries the optimizer should also be able to approximate the size of intermediate results produced by the application of each similarity-based operator. Similarity-based image query processing works in a high dimensional space and the query model used is range query. Therefore, approximation of the size of result of similarity- based operators requires size estimators that consider the dimensionality of the data space and the range query model.

In order to estimate the output of a given algebraic operator we need a means of approximating (knowing) the distribution of the data. Histograms are well known techniques for the approximation of data distribution in statistics. A histogram on attribute is constructed by partitioning the data distribution D into mutually disjoint β subsets called buckets and approximating the frequencies f and values V in each bucket in some common fashion.

Multi-dimensional Histograms

Histograms for the estimation of data distribution in multidimensional data space have been proposed in literature [40]. Gunopulos D. [40] have proposed a new multidimensional histogram technique that is designed to approximate the density of multidimensional datasets. Their technique defines buckets of variable size and allows the buckets to overlap. The size of the cells is based on the local density of the data. The use of overlapping buckets allows a more compact approximation of the data distribution.

Summary

In this chapter we have discussed the data repository model and similarity-based algebra for content-based image database. We have also referred cost model and selectivity estimation functions for similarity-based queries. However, we have not implemented the cost models and selectivity estimation functions. This is because the environment we used for performance testing does not maintain any statistics on the feature vector of images. As a result it become complex to use the cost model and selectivity estimations for the purpose of optimizing similarity-based image queries at this time. Therefore, our work current is purely heuristic-based.

Chapter 4

Query Optimization Method for Content- Based Image Database

In this chapter we will discuss our proposals for query optimization in content-based image database. Hence, a general query optimization framework in the content-based image query processing, proposed query optimization heuristics, an architecture for content-based image query optimizer, and the methods of integrating the query optimizer with existing image DBMS will be presented.

Query optimization in our optimization framework involves two phases of optimization. The first phase of optimization use heuristic (rewriting) rules to transform a query expression involving a similarity-based algebraic expression into an equivalent but more efficient form. This phase of optimization is totally heuristic-based (no cost information is used for transformation).

The second phase of optimization uses implementation rules to map logical expressions chosen by the first phase into an efficient physical algebra or implementation algorithm. As there can be many implementation algorithms for a given operator, this phase of optimization relies on cost estimation of expressions to select an optimal implementation algorithm (physical algebra). The result of this phase of optimization is an optimal query execution plan that is equivalent to submitted query but more efficient in resource utilization. Implementation algorithms are out of the scope of this work.

4.1 **General Query Optimization Framework**

Content-based image query processing follows the same query processing methodology defined for traditional DBMSs. Input queries to the query processor are expressed in declarative languages (such as SQL99, MOQL, FOQL, etc.) which do not requires user knowledge of data representation, access path, or processing strategies.

The parser takes the SQL query expression involving similarity-based operations as an input and parses it to check for synthetic and semantic correctness and then translates it into similarity-based algebraic form. The output of the parser is a similarity-based algebraic expression represented in the form of query tree.

For queries involving similarity-based operations, the query optimizer takes an initial similarity-based algebraic expression as an input and use two kinds of rules to transform them into efficient query execution plan; transformation (logical phase) rule and implementation (physical phase) rules. The output of this phase of optimization can be one or more expressions that involve similarity-based algebraic operations. This is where our similarity-based query optimization heuristics fit. The proposed heuristics will be used to rewrite the input expression into more efficient form. The proposed rewriting rules for similarity-based queries will be presented in the following sections.

The second phase of optimization uses implementation rules to map a given similarity-based algebraic expression into an implementation algorithm or physical algebra (Nested loop, index scan, table scan …). The output of the second phase is an efficient query execution plan that is processed by the query execution engine to answer the query. The execution plan is represented in the form of an operator tree. An operator tree shows the order in which the operators are evaluated and the algorithm used to efficiently implement them. This phase of optimization is based on cost estimation. It takes the statistics(on feature vector of images) required for cost and selectivity estimation from the system catalogue, compute the cost and selectivity of each operator and use this information during transformation of logical expression to efficient execution plan.

The query execution engine takes a query expression in the form of operator tree and processes them according to the orders specified and algorithms selected by the query optimizer to give answer to the input query. It interacts with the database containing images, salient objects and other data to access the data according to the access method chosen by the query optimizer.

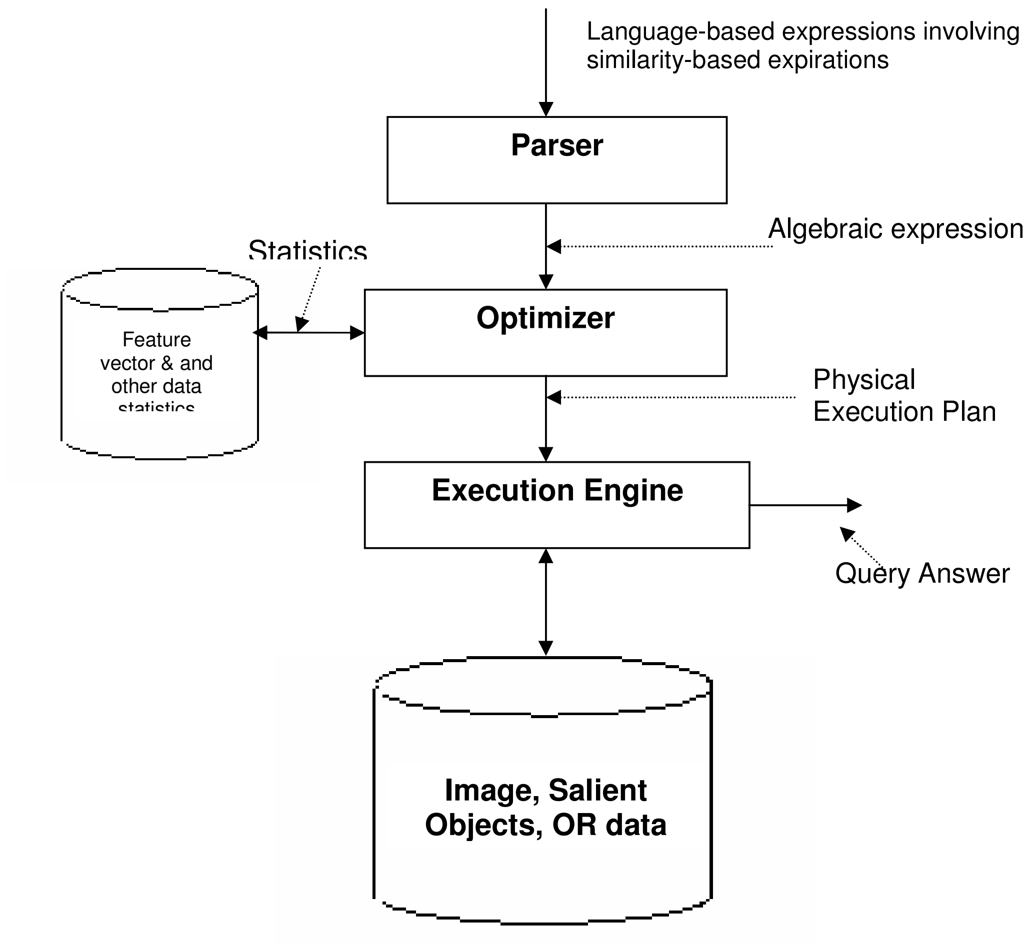


Figure 4-1 General Query Optimization Architecture

The general architecture of query processing and optimization in content-based image database is given in Fig 4-1. An important component in this architecture is the query optimizer. It interacts with the parser, the system catalogue, and the query execution engine. It accepts input query involving similarity-based operations represented in the form of similarity-based algebra. It uses the query transformation heuristics, and cost and selectivity-estimation functions to transform it into optimized form. It interacts with the system catalogue to access statistics on images features for cost and selectivity estimation. Finally it gives an optimized execution plan for the execution engine.

4.2 Query Rewriting Heuristics

An important approach for query optimization technique in our query optimization framework is the transformation of similarity-based algebraic expressions into equivalent but more efficient logical form. The transformation is based on the properties of the similarity-based algebra and their equivalence rules. Unlike general transformation rules (equivalence rules) where transformation is possible both ways, rewriting heuristics proposed in this work are one way. That is, they transform the left hand side (LHS) expression into right hand side (RHS) expressions but the transformation in reverse direction is not optimization. This is what makes them different from general equivalence rules where transformation in both directions is possible. The transformation from LHS to RHS is said to be optimization but the reverse is not. In our rewriting phase, transformations are driven by heuristic rules which are supposed to improve evaluation in terms of speed and memory usage. However, there is no guarantee that they always produce an optimal execution plan for a given expression, but adhering to the following principles turned out to be generally useful.

The proposed query optimization heuristics for the content-based image database are given below.

Rule-1: ***Evaluate relational selection before similarity-based selection.***

$\delta_a (\delta^{\varepsilon}_q (M_1)) \rightarrow \delta^{\varepsilon}_q (\delta_a (M_1))$

Execution of predicates on the metadata component of the image table is cheaper than evaluating similarity-based predicate on the feature vector component. As the attribute predicate reduces the size of data on which similarity matching is applied they have high impact on disk access, memory usage of intermediate results and computation time. Therefore, the relational selection operation is pushed down to the base table so that it is applied before similarity-based selection.

For example, given the SQL query expression below, if no optimization technique is used, the query processor can choose one of the two execution orders or plans: plan1 or plan2.

If the query processor chooses plan1 (Fig 4-2(a)) then it first evaluate similarity-based predicate and the attribute predicate is applied on the data set reduced by similarity-based predicate. As the cost of performing similarity matching is more expensive than evaluating attribute predicate and similarity matching is done possibly on large number of images this plan is not efficient in terms of resource utilization and response time

If on the other hand the query processor chooses Plan2 (Fig 4-2(b)) then it first applies relational selection predicate on all records in the image table and reduce the number of images on which similarity matching is performed. This is an optimal execution plan and can achieve much cost saving depending on the selectivity of relational selection predicate.

Hence this rule uses the knowledge of the existence of predicates on feature vector component and metadata components together in the same query expression to transform the expression to optimal form.

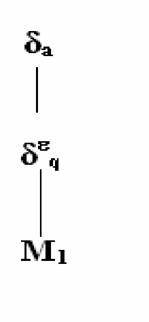
SELECT *OrdSys.IMGScore(12), a.O, a.Path, a.ME_CODE*

FROM M_1 *a ,temp b*

WHERE *ME_CODE<200 and*

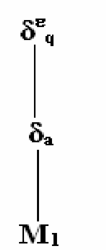
ORDSYS.IMGSimilar(b.F,a.F,'color=1 shape=1 texture =1 location=1' ,20, 12)=1 and b.id=2 order by score

Plan1: $\delta_a (\delta^{\varepsilon}_q (M_1))$



a) Similarity predicate before Relational predicate

Plan2: $\delta^{\varepsilon}_q (\delta_a (M_1))$



b) Relational predicate before similarity predicate

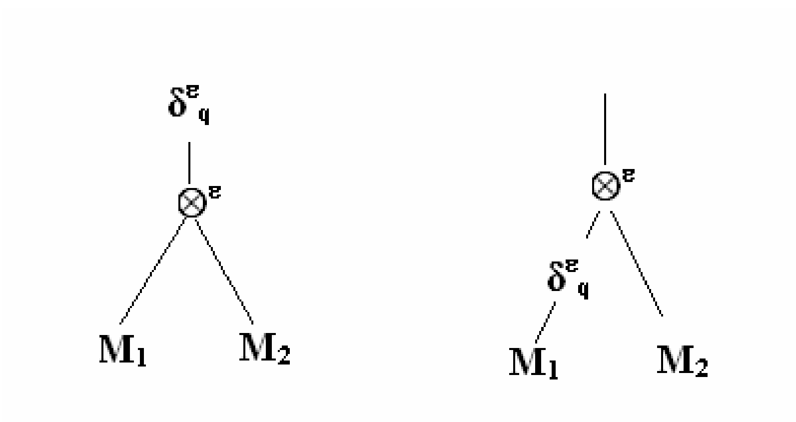
Figure 4-2 Query Tree for order of evaluation of similarity and metadata predicates

Rule-2: ***Perform similarity-based selections before similarity-based join.***

a) $\delta^{\varepsilon}_q (M_1 \otimes^{\varepsilon} M_2) \longrightarrow \delta^{\varepsilon}_q (M_1) \otimes^{\varepsilon} M_2$

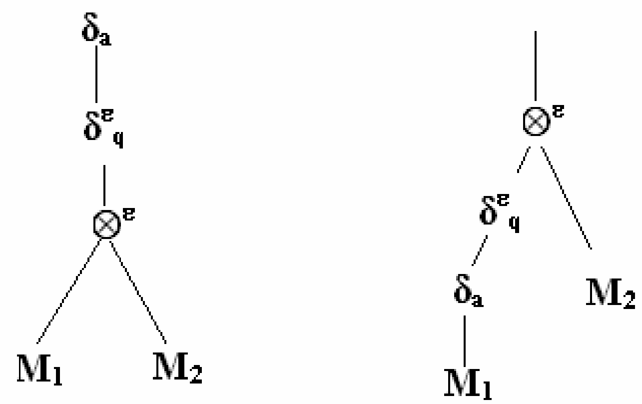
b) $\delta_a (\delta^{\varepsilon}_q (M_1 \otimes^{\varepsilon} M_2)) \longrightarrow \delta^{\varepsilon}_q (\delta_a (M_1)) \otimes^{\varepsilon} M_2$

As in relational query optimization heuristics similarity-based selections are pushed to wards the leaf node of the query tree in order to reduce the result set on which similarity-based join is applied as early as possible. When combined with Rule-1 this rule forces similarity-based selection operator to be applied before similarity-based join operator but after relational selection operator. Fig 4-2(a) shows the pushing down of similarity-based selection. In Fig 4-2(b) is the case where the query expression contains both similarity-based and relational selection, both are pushed down and ordered according Rule-1..



i) $\delta^{\varepsilon}_{q} (M_1 \otimes^{\varepsilon} M_2)$ ii) $\delta^{\varepsilon}_{q} (M_1) \otimes^{\varepsilon} M_2$

a) Query tree for pushing down of similarity-based selection



i) $\delta_a (\delta^{\varepsilon}_q (M_1 \otimes^{\varepsilon} M_2))$ ii) $\delta^{\varepsilon}_q (\delta_a (M_1)) \otimes^{\varepsilon} M_2$

b) Query tree for pushing down of both relational and similarity-based selection predicates.

Figure 4-3 Query tree for pushing selection predicates to base table.

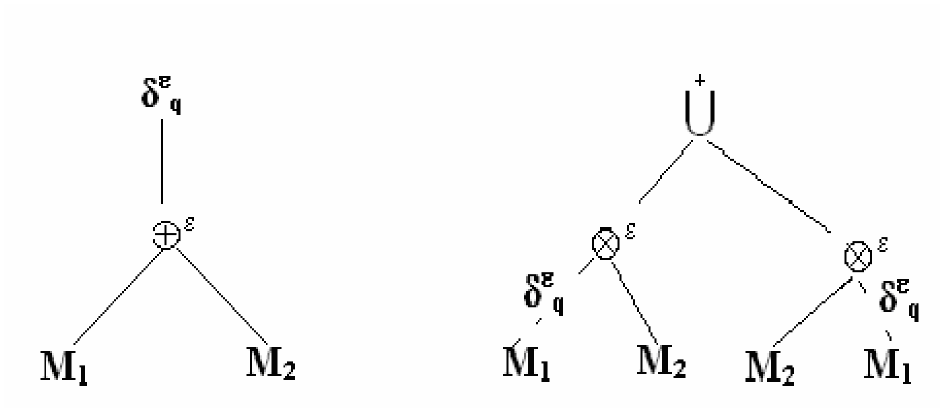
Rule-3: ***Evaluate selection predicates before symmetric similarity-based join operator***

a) $\delta^{\varepsilon}_q (M_1 \oplus^{\varepsilon} M_2) \rightarrow (\delta^{\varepsilon}_q (M_1) \otimes^{\varepsilon} M_2) \cup (\delta^{\varepsilon}_q (M_2) \otimes^{\varepsilon} M_1)$

As it was proposed in [1] after computing the first part of symmetric similarity-based join (i.e. $M_1 \otimes^{\varepsilon} M_2$), the second part (i.e. $M_2 \otimes^{\varepsilon} M_1$) is obtained by applying the *Mine* operator on the result of the first join Mine($M_1 \oplus^{\varepsilon} M_2$). In evaluating the similarity-based join the selection operations can be pushed down to base table to reduce the amount of data on which the similarity-based operation is applied as in the *Rule-2* above.

The application of *Mine* and pushing down the selection operation will reduces the cost of computing the symmetric similarity-based join. This rule can also be extended to optimize the order in which the predicates on feature vector and the metadata component are evaluated.

For example suppose a similarity-based selection operation is defined on symmetric similarity-based join, as show in Fig 4-4(b) the similarity-based selection is pushed down to M_1 so that it is applied to reduce the size of data on which similarity-based joins applied.



a) $\delta^{\varepsilon}_{q}\ (M_1 \oplus^{\varepsilon} M_2)$ b) $(\delta^{\varepsilon}_{q}\ (M_1) \otimes^{\varepsilon} M_2)\ U\ (\delta^{\varepsilon}_{q}\ (M_2) \otimes^{\varepsilon} M_1)$

Figure 4-4 Pushing Similarity-based selection in symmetric similarity-based join

Rule-4: ***Avoid redundant evaluation of predicates on the left most table of multi similarity-based join.***

a) $\delta^{\varepsilon}_{q}\ (M_1 \otimes^{\varepsilon} M_2 \otimes^{\varepsilon} \ldots \otimes^{\varepsilon} M_n) \longrightarrow (M'_1 \otimes^{\varepsilon} M_2)U\ (M'_1 \otimes^{\varepsilon} M_3)U \ldots U\ (M'_1 \otimes^{\varepsilon} M_n)$

b) $\delta_a\ (\delta^{\varepsilon}_{q}\ (M_1 \otimes^{\varepsilon} M_2 \otimes^{\varepsilon} \ldots \otimes^{\varepsilon} M_n) \longrightarrow (M''_1 \otimes^{\varepsilon} M_2)U\ (M''_1 \otimes^{\varepsilon} M_3)U \ldots U(M''_1 \otimes^{\varepsilon} M_n)$

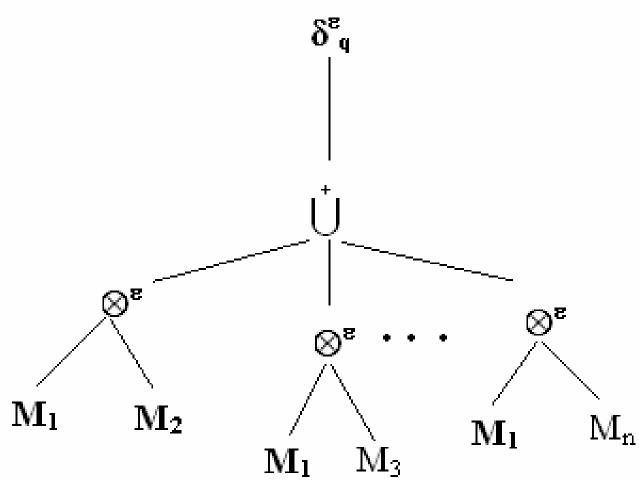
Where $M'_1 = \delta^{\varepsilon}_{q}\ (M_1)$ and $M''_1 = \delta^{\varepsilon}_{q}\ (\delta_a\ (M_1))$

In query expressions involving multi similarity-based join operation the left most table is joined with all other tables in the expressions as an outer join and the result of pair wise joins are merged by additive union operator. If selection predicate (similarity-based) exists on the left most table of the multi similarity- based join then it is applied for each pairwise joins. We propose a heuristic to apply selection predicates only once and perform pairwise similarity-based joins on the reduced intermediate table. This heuristic avoids the cost of redundant application of the selection predicates on M_1 component of the query expression and the cost of managing intermediate results.

Multi similarity-based join is defined as:

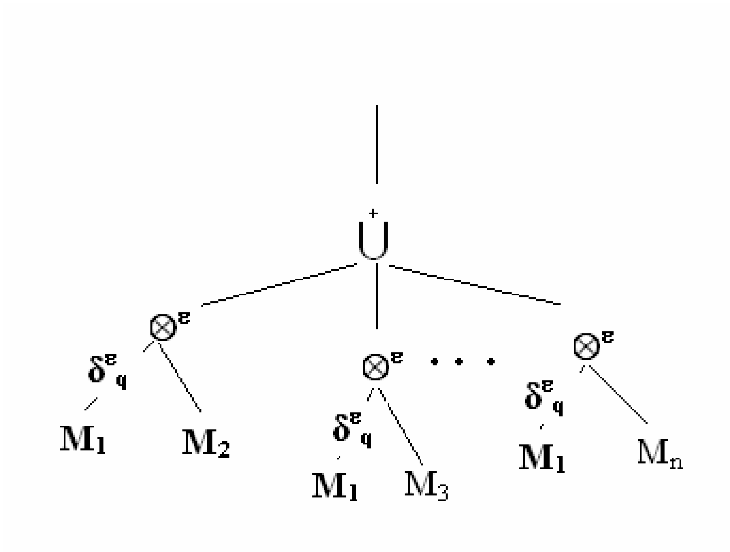
$$M_1 \otimes^{\varepsilon} M_2 \otimes^{\varepsilon} \ldots. \otimes^{\varepsilon} M_n = M_1 \otimes^{\varepsilon} M_2 \cup M_1 \otimes^{\varepsilon} M_3 \cup \ldots \cup M_1 \otimes^{\varepsilon} M_n$$

As the pairwise join operations are performed on reduced data sets and the predicates are applied only once instead of being applied many times, the rule may result in considerable reduction of query evaluation time. Fig 4-5(c) shows the optimized form of algebraic expression given in Fig 4-5(a) according to Rule-4.



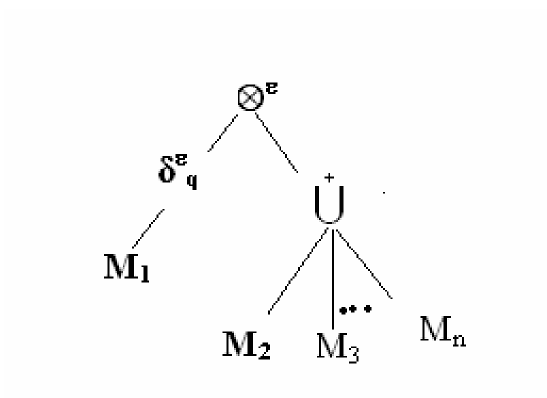
$$\delta^{\varepsilon}_{q} (M_1 \otimes^{\varepsilon} M_2 \otimes^{\varepsilon} \ldots. \otimes^{\varepsilon} M_n) = \delta^{\varepsilon}_{q} (M_1 \otimes^{\varepsilon} M_2 \cup M_1 \otimes^{\varepsilon} M_3 \cup \ldots \cup M_1 \otimes^{\varepsilon} M_n))$$

a) Multi similarity-based join with selection predicate on M_1 not pushed down



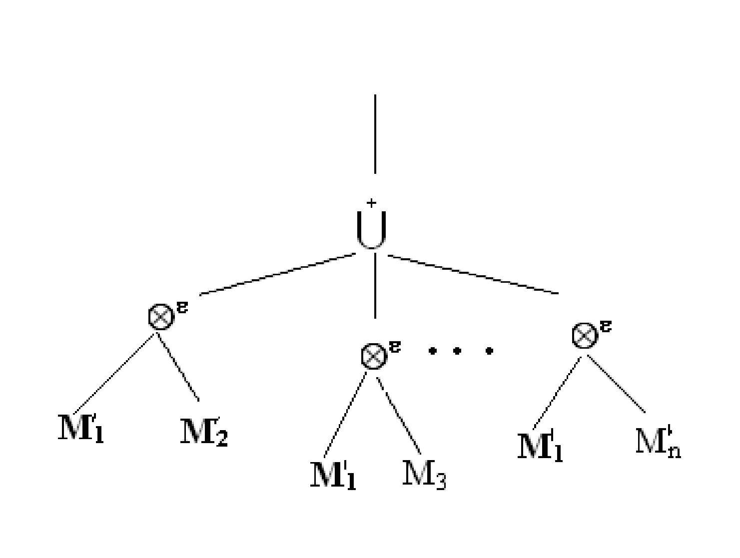
$$\delta^{\varepsilon}_{q}\ (\delta_{a}\ (M_1))\otimes^{\varepsilon} M_2 \cup \delta^{\varepsilon}_{q}\ (\delta_{a}\ (M_1)) \otimes^{\varepsilon} M_3 \cup \ldots \cup \delta^{\varepsilon}_{q}\ (\delta_{a}\ (M_1)) \otimes^{\varepsilon} M_n$$

b) Multi similarity-based join with selection predicate pushed down to M_1



$$\delta^{\varepsilon}_{q}\ (\delta_{a}\ (M_1))\otimes^{\varepsilon} (M_2 \cup M_3 \cup \ldots \cup M_n)$$

c) Multi similarity-based join with selection predicate pushed down to M_1 and redundant computation of its selection is eliminated



d) Multi similarity-based join with redundant computation is avoided (M'_1) is an intermediate table obtained by performing selection predicates on M_1)

Figure 4-5 Avoiding redundant predicate evaluation on multi similarity-based join

Rule-5: ***Avoid redundant evaluation of predicates on the same table when performing Symmetric multi similarity-based join.***

$\delta^{\varepsilon}_{q} (M_1 \oplus^{\varepsilon} M_2 \oplus^{\varepsilon} .... \oplus^{\varepsilon} M_n)$

$\equiv \delta^{\varepsilon}_{q} ((M_1 \otimes^{\varepsilon} M_2) \text{ U } (M_2 \otimes^{\varepsilon} M_1) \text{ U } (M_2 \otimes^{\varepsilon} M_3) \text{U } (M_3 \otimes^{\varepsilon} M_2))\text{U}...M_n \otimes^{\varepsilon} M_{n-1}$

$\equiv (\delta^{\varepsilon}_{q} M_1) \otimes^{\varepsilon} M_2 \text{ U } (\delta^{\varepsilon}_{q} M_2) \otimes^{\varepsilon} M_1 \text{ U } (\delta^{\varepsilon}_{q} M_2) \otimes^{\varepsilon} M_3 \text{ U ...U } \delta^{\varepsilon}_{q} (M_n) \otimes^{\varepsilon} M_{n-1}$

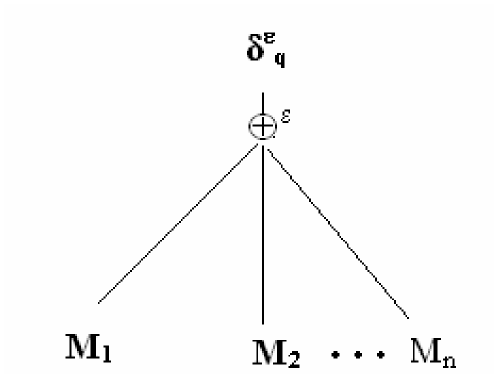
$\rightarrow (M'_1 \otimes^{\varepsilon} M_2) \text{ U } (M'_2 \otimes^{\varepsilon} M_1) \text{ U } (M'_2 \otimes^{\varepsilon} M_3)\text{U } (M'_3 \otimes^{\varepsilon} M_2)\text{U... } M'_n \otimes^{\varepsilon} M_{n-1}$

Simple push down of selection predicate to base tables during the evaluation of symmetric similarity-based join results in redundant evaluation of the predicates. To avoid the redundant application of predicates on the same table the predicate is applied only once and the actual similarity-based joins are applied on the reduced intermediate table. The algorithm here is to apply selection predicates for each table only once and use the reduced intermediate table when performing join. For example, suppose we have a similarity-based selection on the table M_2 in a multi similarity-based join involving n tables then since M_2 appears four times in the expression the predicate is evaluated four times.

The symmetric multi similarity-based join of n tables is given by:

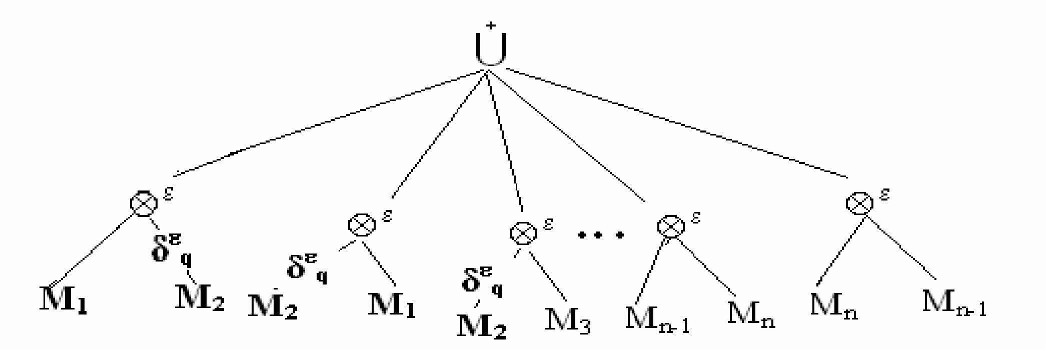
$(M_1 \oplus^{\varepsilon} M_2 \oplus^{\varepsilon} \dots \oplus^{\varepsilon} M_n) =$

$(M_1 \otimes^{\varepsilon} M_2)$ **U** $(M_2 \otimes^{\varepsilon} M_1)$ **U** $(M_2 \otimes^{\varepsilon} M_3)$ **U** $(M_3 \otimes^{\varepsilon} M_2)$ **U**...**U** $(M_n \otimes^{\varepsilon} M_{n-1})$



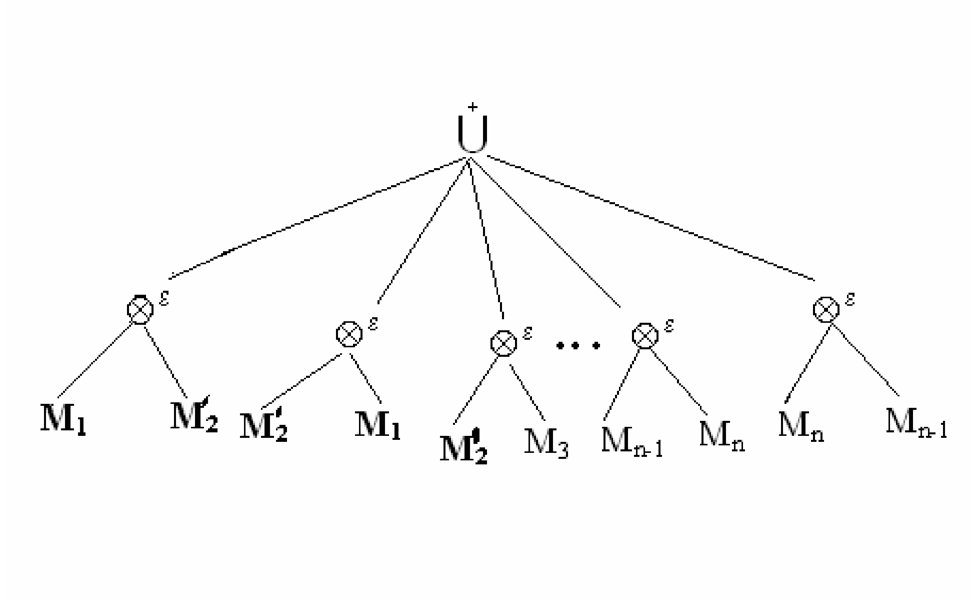
$\delta^{\varepsilon}_q (M_1 \oplus^{\varepsilon} M_2 \oplus^{\varepsilon} \dots \oplus^{\varepsilon} M_n)$

a) Symmetric multi Similarity-based join with selection predicate not pushed down



$M_1 \otimes^{\varepsilon} (\delta^{\varepsilon}_q M_2) \cup (\delta^{\varepsilon}_q M_2) \otimes^{\varepsilon} M_1 \cup (\delta^{\varepsilon}_q M_2) \otimes^{\varepsilon} M_3 \cup \dots \cup M_n \otimes^{\varepsilon} M_{n-1}$

b) Symmetric multi Similarity-based join with selection predicate pushed down predicate on M_2 is evaluated twice



$(M'_1 \otimes^{\varepsilon} M_2) \cup (M'_2 \otimes^{\varepsilon} M_1) \cup (M'_2 \otimes^{\varepsilon} M_3) \cup (M'_3 \otimes^{\varepsilon} M_2) \cup \ldots M'_n \otimes^{\varepsilon} M_{n-1}$

c) Symmetric multi Similarity-based join redundant evaluation of Predicate is eliminated M'_2 is an intermediate table

Figure 4-6 single evaluation of predicates on base tables of Symmetric similarity-based join

Rule-6: *Unnesting Nested Queries*

Nested queries have been extensively studied in the area of traditional database systems. Nested queries contain more than one selection operations in one query expression. In nested query expressions, for each record of the outer selection the inner selection is performed once. One solution proposed for the optimization of nested queries is unnesting (or Decorrelation). Unnesting transform the multiple selections blocks into joins. Once it is transformed into flat form different optimization techniques defined for joins and the ordering of selection predicates proposed in Rule-1 can be easily applied.

For example query of the form: Display all images similarly to a given query image from a table M1 that have similar image in M_2 that fall in a given category and taken in a given time period.

This query can be expressed in SQL in the first box of Fig 4-8 and its flattened form is given in the second box.

```
SELECT        O,  A, P
FROM M1 m,
WHERE         m.Similar(q, ε)
AND    m.F similar(
          SELECT n.F
          FROM M2 n
                    WHERE n.A.Photodatae <??
                    AND n.A.Category=??, k )
```

```
SELECT      id, O, A, P
FROM        M1 m, M2 n
WHERE       m.Similar(q, ε)
AND    m.F ≈k n.F
AND    n.A.Photodatae <??
AND    n.A.Category=??
```

Figure 4-7 Example of Unnesting Nested similarity-based query

Rule-7: ***Use index on the F component of an image table.***

In traditional database systems the role of indexing for efficiency of query processing is well understood. As a result almost all known DBMSs include one or more indexing schemes that can be used to index attribute data. Indexes limits the data retrieved from disks and the comparisons made. This results in reduced I/O operation and computation time.

As in traditional database systems, for large image databases, sequential scanning of the whole pages is computationally infeasible. To search large content-based image databases, indexing of the feature vector to accelerate the retrieval has become more important and even a must. For content-based image retrieval indexing structures are based on representing the feature vector as points in a multi dimensional vector.

Like the B-tree indexing used in traditional database systems, structurally, most multidimensional indexing structures are based on hierarchical clustering of image features. However, these multidimensional indexing structures are different from B-tree and other indexing for traditional data in that the internal node of the tree represents n-dimensional hyperrectangles (for R-tree and X-tree and R*-trees) rather than a scalar range.

For indexing of image features, most existing content-based image database systems allow the definition of more than one index types on the same column containing image feature. There are also proposals for building index structures on fly when the query is processed.

The heuristic we propose is to use one of the existing indexing structures on the feature vector component of the image table. The decision of which index to use and weather to build index on fly should be based on some cost analysis.

Rule-8: ***Index Scan the base relational on metadata component.***

In multi-criteria queries on content-based image database, the attribute information is used in conjunction with similarity-based operations. While accessing the base table, if the relational selection operation is chosen to be applied first according to Rule-1 then a B-tree index defined on the column on which the predicate is applied is used.

4.3 Integrating Content-based Image Query Optimizer into Image DBMS

There are two options for the integration of content-based image query optimizer into existing OR database systems. The first option is to extend the existing query optimizer by introducing statistics collection functions, similarity-based algebra, transformation rules and the cost model. However, most of the existing OR database systems do not allow the extension of the rule-based optimizer(i.e. the introduction of query optimization heuristics and transformation rules developed for content-based image queries based on similarity-based algebra is not an easy task). They only allow extension of the cost-based optimizer. Oracle for example provides extensibility for the cost-based optimizer. Introduction of operators, statistics collection functions, cost estimation function, and selectivity estimation functions can be easily achieved. But, the rule-based optimizer can not be extended. Hence, it is not an easy task to introduce optimization heuristics proposed for similarity-based algebra and full advantage of the query optimizer can not be taken for content-based query processing. This forces us to look for an alternative way of integrating query optimizer and query processing for content-based query processing.

The second opinion is to add query optimizer as an additional module in the existing system. In this regard S.Atnafu et.al. [48] have proposed general architecture (Fig 4.8) for image DBMS that work with existing Object relational DBMS. Their architecture extends the existing ORDBMS systems to support content-based image queries. They have added three modules (image query processor, image and salient object routines and data repository manager) that are important for the extension of content-based image query processing.

The central component of the architecture is the Image Query Processor (IQP). IQP is proposed to extend image query processing operations. It is proposed to consist of similarity-based operators, the image query processing engine and the content-based query optimizer.

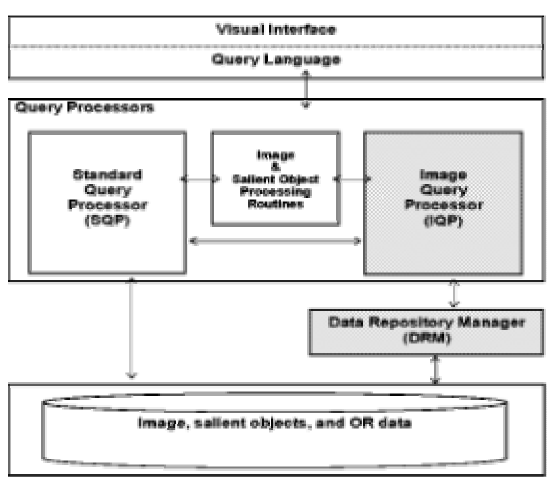


Figure 4-8 General Architecture of an Image DBMS [48].

Considering the problem that exists with the current commercial database management systems that support content-based image query processing to fully extend their optimizer we follow the general architecture proposed in [48]. Hence, the query optimizer proposed in this work will be integrated into the IQP of the general architecture of image database. In this query processing architecture the query is expressed using visual interface which then translated into query language used. The query optimizer accepts expressions translated into similarity-based form and transforms it into an efficient execution plan using heuristic rules and cost information. The optimizer optimizes the query by applying rewriting rules proposed in this thesis and cost estimation functions proposed in this work. The query optimizer produces an efficient query execution plan which is then executed by the execution engine to give answer.

The interaction of the proposed query optimizer with other components of the database system is given in Fig 4-9.

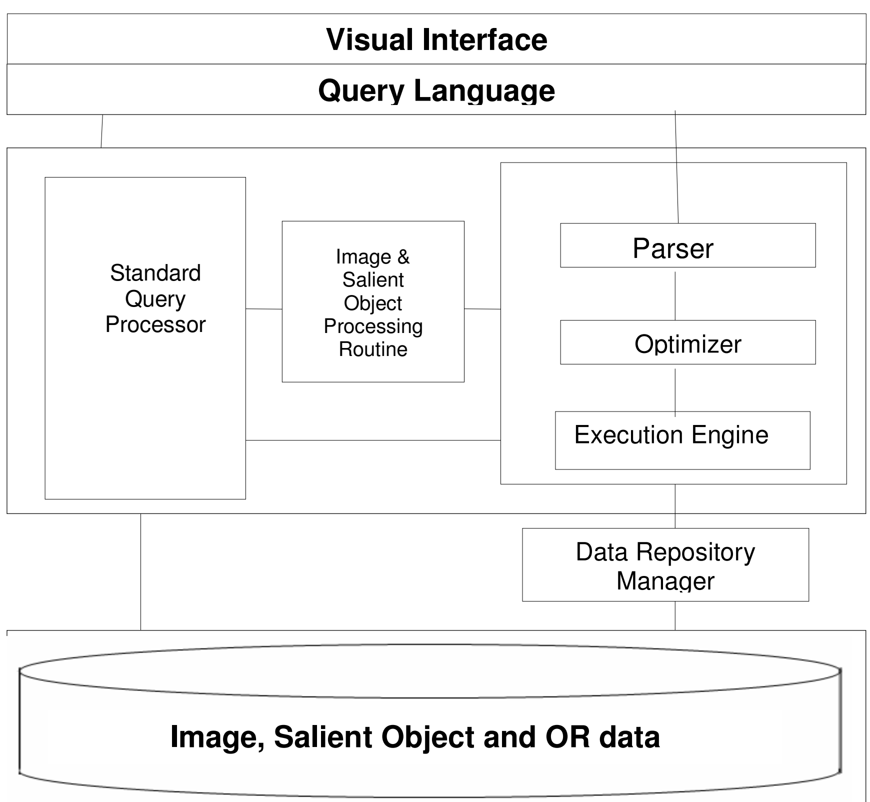


Figure 4-9 General Architecture Showing Integration of Query Optimizer with Image database

4.4 **Modeling a query optimizer for content-based image database**

Query optimization in traditional database systems has been studied for the last 30 years. As a result many query optimizers that implement the query optimization techniques defined for these database systems have been proposed and implemented. Building an efficient query optimizer however, is still an active research topic.

Content-based image database is based on a different query evaluation strategy; query evaluation is based on degree of similarity between images based on their feature vector rather than exact match, Cost models and access methods (B-tree index) designed for attribute data are not efficient for content-based image query evaluation, and internal query processing is using the similarity-based algebra. Due to these differences in query evaluation relational query optimizers do not adders the problem of query optimization for content-based image query optimization.

Most of the query optimizers designed for commercial OR database systems even though they are theoretically extensible, they are not fully extensible. Extension of the rule-based optimizer by introducing additional transformation rules is not an easy task.

Motivated by these problems, we have designed a query optimizer that meets requirements of content-based image database.

In this section we will discuss the requirements of query optimizer for content-based image database, and its proposed design.

4.4.1 **Requirements**

Three important requirements for the query optimizer for content-based image database are identified. These are:

Correctness: The optimizer should produce a correct execution plan. In other words, the optimized query expression should give the same result with the initial query submitted.

Reduced runtime overhead: The optimizer should not spend much time to look for an optimal execution plan.

Extensibility: The optimizer should be extensible

> *"Extensibility is the process of augmenting or removing features easily from a system in order to customize it for an application. Extensibility of query optimizers refers to the ease of constructing optimizers for extensible DBMSs. It also refers to the easy customizability of an existing query optimizer to a new application." [28]*

Extensibility required for the following reasons:

- Query processing in content-based image database is an emerging research area. As a result, additional algebraic operators and additional transformation rules may be identified and the existing ones may be changed,
- Algebra, transformation rules, and cost function for salient objects may need to be included into the query optimizer.
- Implementation algorithms for the realization of the proposed algebra including join methods, sorting, etc. may be identified.
- Better cost models and selectivity functions may be derived.
- New access methods (e.g. indexing structures) may be identified.
- New query optimization techniques may be identified and the existing ones may be changed.
- Current content-based image databases do not maintain statistics on the feature vector of images. In the future when this information is maintained it is likely that many cost based optimizations will be proposed.

To meet these requirements the optimizer should be easily modifiable. To achieve these goals, the optimizer should comprise of an independent subsystem that performs optimization operations and provides a certain interface for using these operations.

4.4.2 Design of Content-based image Query Optimizer

Based on the requirements listed above and considering a heuristic-based query optimization techniques we proposed in the previous section, a rule-based and extensible query optimizer is designed. The query design of optimizer extensible for addition and modification of heuristic rules and cost functions and search strategy.

The design of the proposed query optimizer is based on a pattern proposed in DORS [13] extensile rule-based query optimizer. It is adapted to meet the requirements of query optimization in content-based image database.

Query optimizers have three important components; the search space, the cost model and a search strategy. The class diagram in Fig 4-10 shows the structure of the query optimizer for content-based image database.

The optimizer uses two optimization phases; logical and physical. The logical optimization phase uses similarity-based algebraic rewriting rules to transform a given logical expression into an equivalent and most efficient form. The physical phase of optimization uses implementation rules to map the output of the logical phase of optimization to an efficient implementation algorithm.

In Fig 4-10 CBIDBMOptimizer class acts as an interface to query processor. It is called by the query processor with the algebraic expression as an argument. It is an abstract class which is implemented by CBIDBOptimizer. CBIDBMOptimizer uses the CBIDBRuleFactory class to create rules. The CBIDBRuleFactory intern uses the physicalCBIDBRuleFactory and logicalCBIDBRuleFactory which creates and returns logical and physical rules applicable for the expression given respectively. The CBIDBRule is an abstract class which defines methods implemented by each concrete Rules. CBIDBOptimizer uses the CBIDBSearchStrategyFactory class to create the search strategy which takes the rules and the expression as an argument to process each rule received from the rule factory against the expression.

Representation of Rules

Rules are defined by CBIDBRule abstract class. Each heuristic rules and implementation rules extend and implement this abstract class. Each rule is composed of four important components; a header, a promise value, an applicability condition and transformation code.

Header: The header of rules defines a pattern matching description of the target expression.

The applicability condition: The applicability condition determines if the rule i\should be fired or not.

The transformation code: is a code that performs the transformation if the applicability condition returns true.

*The promise***:** The promise value is used to order the application of the rule. When more than one rule is found to be applicable for a given expression, the promise value determines the order of application of these rules.

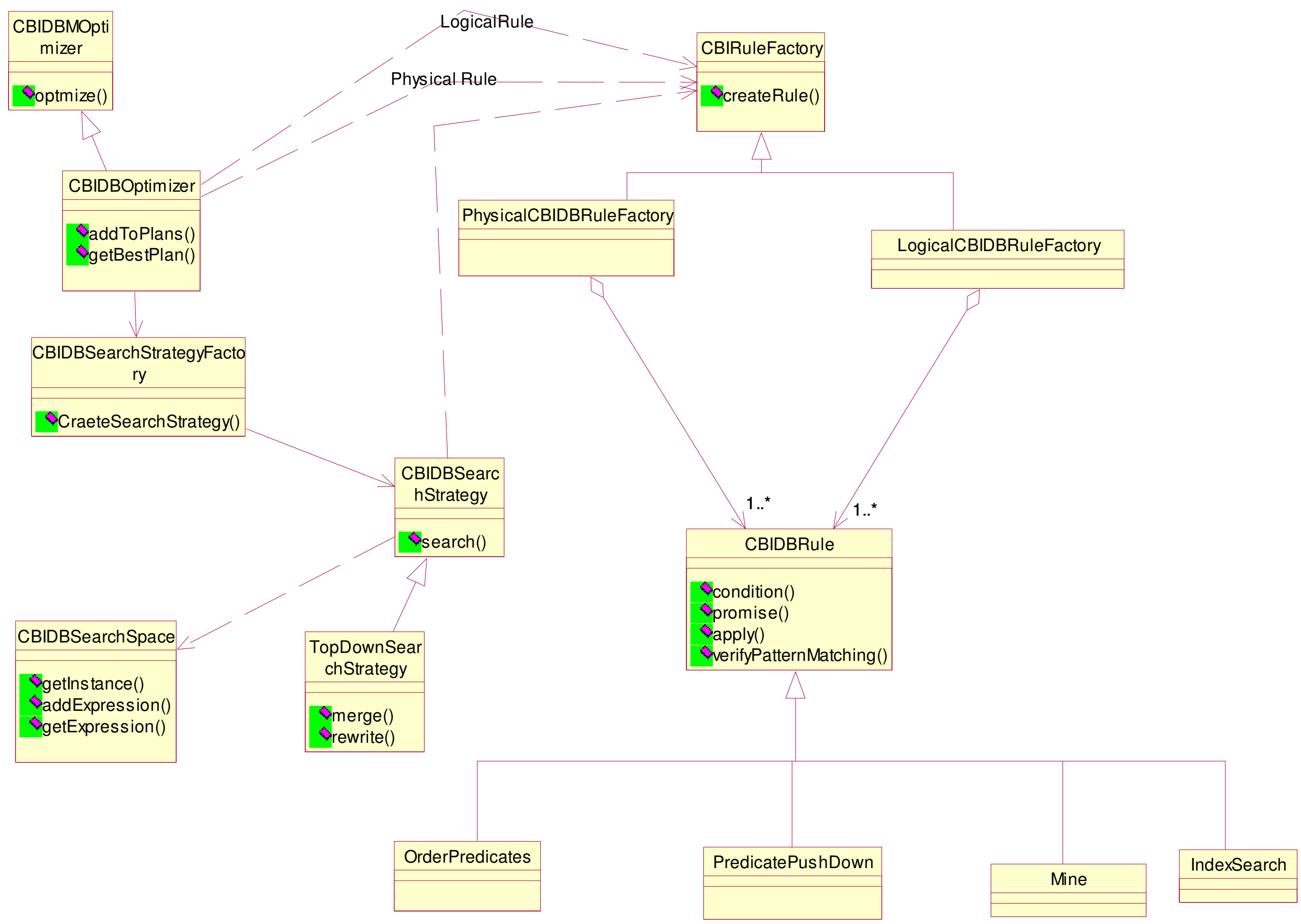


Figure 4-10 Object Oriented Design of the CBIDBQuery Optimizer

Specific Scenario

A specific scenario which shows the dynamics of the CBIDBOptimizer is given in Fig4-11.

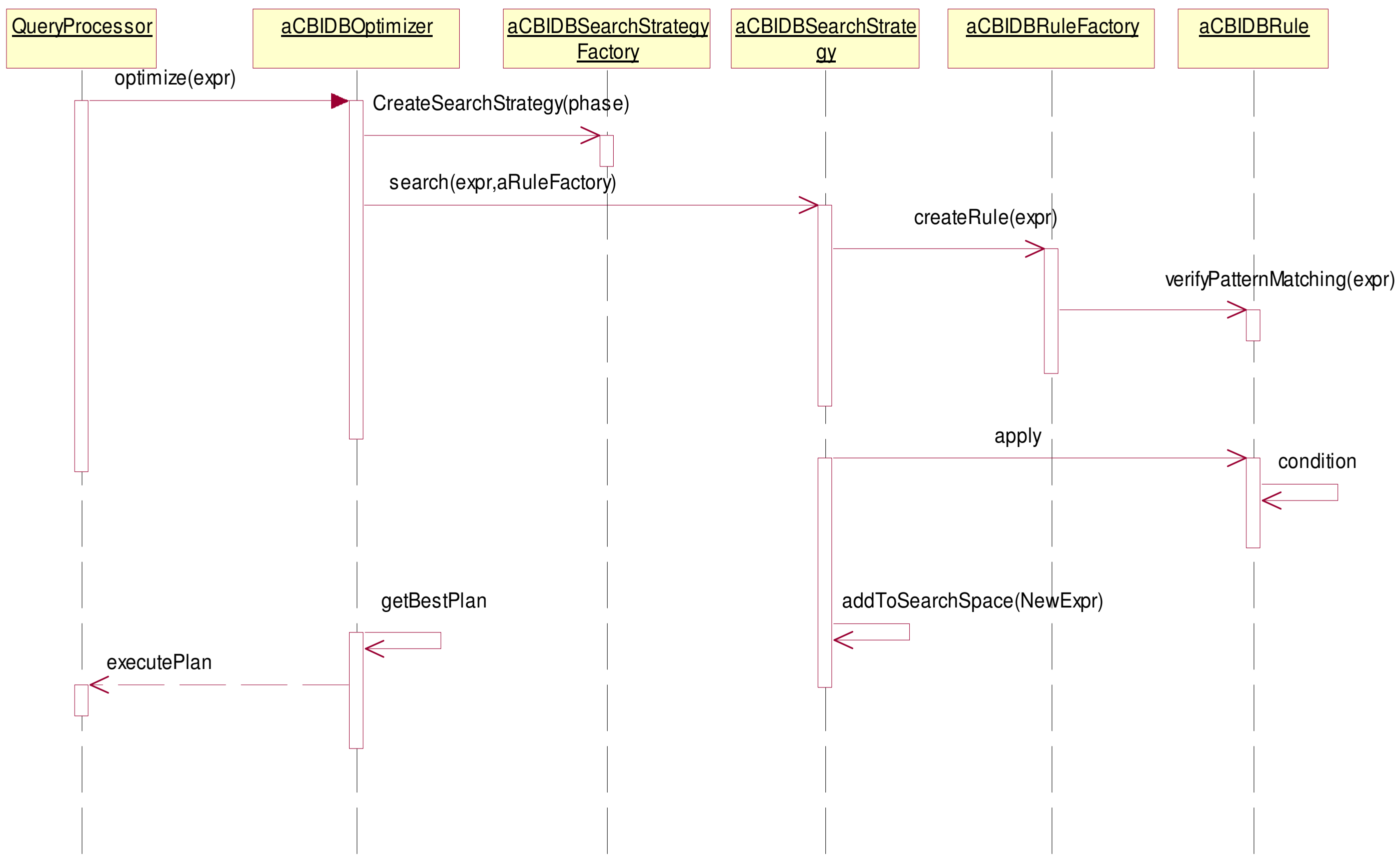


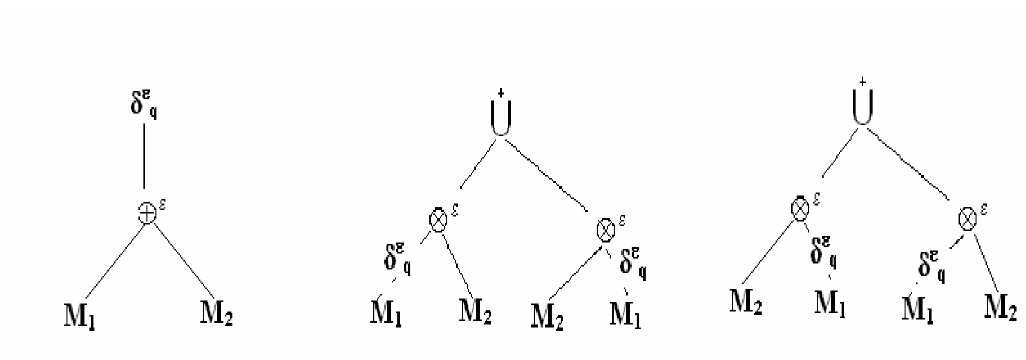
Figure 4-11 Scenario of query optimizer

4.5 **Components of the Query optimizer**

4.5.1 **Search space**

Similarity-Based queries are internally represented using similarity-Based algebra as an intermediate representation language. The abstract representation of the query is using query tree. Query trees are used to describe the order in which the operators are implemented. The leaf nodes of the query tree represent the base relations. The non leaf nodes are similarity-based and relational algebra operators such as similarity-based selection, relational selection, similarity-based join etc. The edge of the tree represents data flow from bottom up. An intermediate node of the tree indicates the application of the corresponding operator on the relations generated by its children and its result is further sent up the tree. That is from the leaves which correspond to data in the database to the root which is the final operator producing the query answer for the query.

As the symmetric similarity-based join has many set theoretic properties such as commutatively, associatively, and distributativity, there can be large number of equivalent query trees representing the same query expression. For example, given the expression, σ^{ε}_{x} ($M_1 \oplus^{\varepsilon}$ (M_2)) we may have the following equivalent query trees with varying cost of evaluation.



a). σ^{ε}_{x} ($M_1 \oplus^{\varepsilon} M_2$) b). (($\sigma^{\varepsilon}_{x}$ M_1) $\otimes^{\varepsilon}$ M_2)U (M_2 $\otimes^{\varepsilon}$ σ^{ε}_{x} (M_1)) c). (M_2 $\otimes^{\varepsilon}$ σ^{ε}_{x} (M_1)) U((σ^{ε}_{x} M_1) $\otimes^{\varepsilon}$ M_2)

Figure 4-12 query three for different ways in which the expression can be evaluated

For complex query expressions that involve large number of image tables, the number of equivalent expressions may be extremely large. Hence the query optimizer has to be equipped with appropriate search strategy.

Size of search space

In one way Similarity-based join the order of relations have semantic meaning. The join operations ($M1 \otimes^{k} M_2$) and ($M_2 \otimes^{k} M_1$) do not give the same result. However, each one way similarity-based join can be evaluated in two ways, by itself or using the *Mine* operator depending on the cost estimate. The search space for query expression involving N image tables will have (N-1) pair wise joins where each pair wise join is implemented in two different ways resulting in $2^{(N-1)}$ different possible joins.

Symmetric similarity-based join is commutative that is the order in which the relations are evaluated has no effect on the query result. Moreover, each such join is implemented by two one way similarity-based joins. These two properties make the number of different alternatives of evaluating the query expression very large. For example, considering the commutativity property of the symmetric similarity- based join, for N table queries the selection of inner and outer tales generates $2^{(N-1)}$ equivalent join trees. In this case the search space consists of at least $2^{(N-1)}$ trees. When the application of the *Mine* operator is considered the number of alternatives considered will increase to a very large number resulting in a large search space.

At the implementation stage each of the logical query trees may be mapped to many equivalent operator trees for their implementation. For example, a join operation can be implemented using simple nested loop join implementation, hash join etc. and the selection operation can be implemented using sequential scan or index scan depending on the size of estimated query result (selectivity of the predicates). This will result in an increase of the search space to very large size.

The drawback of the large search space is that the query optimizer will spend much time searching for an optimal query plan than processing the query (In other words there will be higher runtime overhead for the query processor). In query optimization it is favorable that the running time for the search of optimal execution plan is minimized. A common way to do this is by the application of Heuristics for pruning of those trees that are not considered to produce optimal plan.

4.5.2 **Search strategy**

A search is another important component of query optimizers. It is an algorithm used to search an optimal plan in the search space. As discussed in the previous section the search space for content-based image queries is large and there is a need to employ an appropriate search strategy.

Dynamic programming which is used in most of commercial and experimental systems can be adapted for this purpose. Considering the efficiency of the strategy the top down strategy which is implemented in Volcano optimizer for the first time is proposed as an important candidate for searching an optimal execution plan. The top down strategy is implemented in many query optimizers such as TIGUKAT [42] project for Objectbase database and proven to be efficient.

Representation of Search Strategy in the query optimizer

The search strategy is defined using an abstract class. Each search strategy implemented will implement a concrete class of the CBIDBSearchStrategy. The class diagram in Fig 4-10 shows a TopDownsearchStrategy implementing CBIDBSearchStrategy abstract class. This enables us to add and modify search strategies and more than one search strategies can be implemented from which the optimizer chooses during optimization. During query optimization the CBIDBOptimizer class the CBIDBRuleFactory class to create an appropriate search strategy.

4.5.3 **Cost Model**

Cost model is another important component of query optimizers. Cost models estimate the cost of processing a given algebraic operator. In addition to cost estimation function, the selectivity functions that estimate the output of a given algebraic operator are also important components of the cost model. The cost model and selectivity function are discussed in chapter three.

Representation of Cost Model in the query optimizer

In the query optimizer, the cost model is defined in terms of an abstract method in the CBIDBRule class. Each physical operator will implement the algorithm for estimation of cost of algebraic operator.

Summary

In this chapter we have presented the general query processing framework for content-based image database. We have proposed query optimizations heuristics, identified requirements of query optimizer for content-based image database and identified the design of the proposed query optimize. We have also proposed a method of integrating the query optimizer with existing image database systems. In addition we have reviewed the cost model and the search space of query optimization.

However, the design of the query optimizer is not implemented. This is because the environment we choose for experimentation is a commercial database system. As a result we could not modify query optimizer of the system as this is not easy for closed commercial systems.

On the other hand, Query optimizers of open systems such as mySQL, Postgress etc. are modifiable. These systems allow one to extend the system to provide new functionality and integrate components including the query optimizers. We did not choose these systems because they do not have an image feature extraction engine which is used to extract image features and represent them in a database.

Chapter 5

Experimental results

5.1 Experimental Setting

Some performance tests were made to study the effects of the optimizations heuristics proposed. The tests evaluate computation cost by measuring the execution time of the queries. The experiment is performed on Intel Pentium-4 computer with 2.0GHZ speed and 256MB main memory running windows 200 advanced server. Oracle9i version 9.0.1 server is installed on the server machine. The rules are implemented in java using JDBC interface. The database is stored on the server with the above specification and the query is executed from a client machine with the same hardware specification. On the client machine oracle client application is installed. For each test the query is executed 10 times and an average value is taken. To avoid the biases due to very large numbers that may occur when the system may be busy serving other processes extremely large values are discarded from computation of average.

Each test query is performed for two cases, one with and the other without the query rewriting rules applied and the execution times of the query are measured.

The schema of the table used for testing is of the form:

$$M(id,A,O,F,P)$$

The query of *Experiment-1* includes ordering similarity and relational selections(*Rule-1*). The query of *experiment-2* is performed to test the performance of *Rule-2*. *Experiment-3* is performed to test *Rule-3*. The query of *Experiment-4* to test performance of accessing base table using indexing structure on the feature vector component of image table.

5.2 Experimental Tests

In this section we will report the result of experimental results of proposed optimization techniques.

Experiment-1: Performing Relational selection predicate before Similarity-based selection predicate (Rule-1)

The experiment is performed with query of the form:

*SELECT a.**

FROM M_3 a, temp b

WHERE ME_CODE<50 and ORDSYS.IMGSimilar(b.F,a.F,'color=1 shape=1 location=1 texture=1', 20,123)=1 and b.id=2

This is algebraically expressed as:

$$(\delta_{me\_code<50}\delta_{IMGSimilar(b.F,a.F,,20})(M_3))$$

This algebraic expression can be processed in one of the two possible execution plans which have different costs. According to Rule-1 Plan2 is optimal.

Plan1: $(\delta_{me\_code<50}\delta_{IMGSimilar(b.F,a.F,,20})(M_3))$

Plan2: $(\delta_{IMGSimilar(b.F,a.F,,20}(\delta_{me\_code<50}(M_3))$

The experiment is performed on four tables of sizes varying from 50 images to 1000 images as shown in Table-1 below.

Each test is performed 10 times and an average value is taken. The query is executed one with the similarity and relational selections ordered according to the heuristic and the other with out applying the optimization heuristic.

The results are shown in Table1 and Fig5-1. Form the table and the chart we see that there is a substantial cost saving in terms of execution time by the application of the heuristic rule. Cost saving is achieved because similarity-based selection operation is made to execute less number of images that are reduced by applying the relational selection first. The optimization is sounder when the size of table becomes larger. This is because the on a large sized table it is more likely that the relational selection is more selective than on table with smaller size.

Table 5-1 Result of ordering similarity-based and relational selection predicates

No of Records in the table	After Rewriting (millisecond)	Before Rewriting (millisecond)
50	22	113
100	61	217
500	108	376
1000	116	498

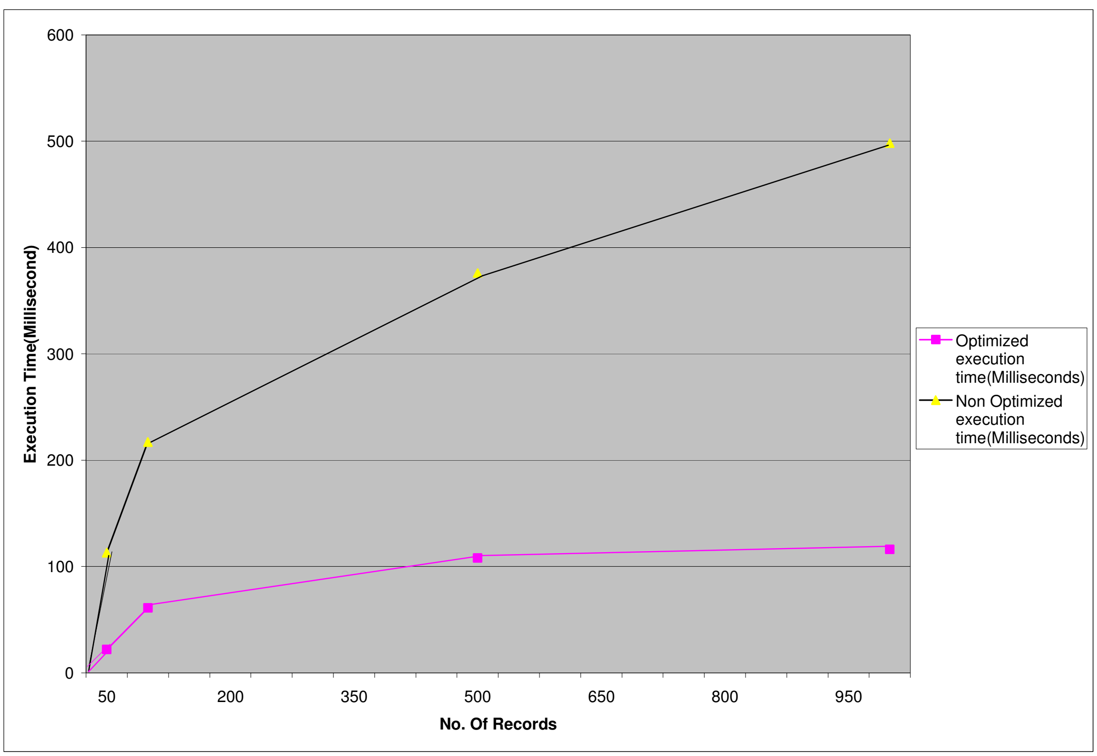


Figure 5-1 Result of Experiment on Ordering similarity-based & relational selections

The following table and chart shows the performance testing of Rule-1 with different selectivities of the relational selection. Selectivity is the ratio of recods selected by the selection predicate. the result shows that with the increase in selectivity there is more performance gain in terms of execution time.

Selectivity= $Number-of\_records\_selected \Big/ Total\_No\_of\_records$

Table 2 result of rule-1 with different selectivities of relational selection

Selectivity of Relational selection	Non Optimized	Optimized
0.25	183	47
0.5	185	78
0.75	186	120
1	187	187

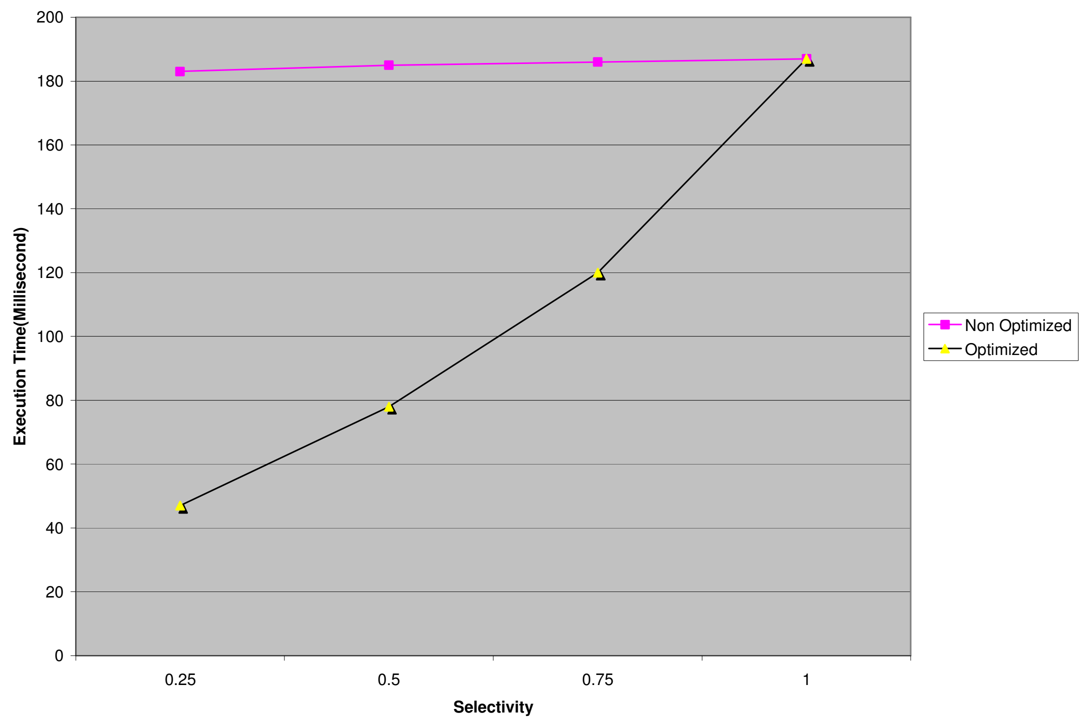


Figure 5-2 result of rule-1 with different selectivities of relational selection

Experiment-2: Performing similarity-based selection before similarity-based join

The second experiment is performed to test the performance of Rule-2, the rule of pushing similarity-based selection to base table. The test is performed on query of the form:

*SELECT a.**

FROM M_1 a, M_3 b, temp c

WHERE SimJoin(a.F, b.F, ε) and

ORDSYS.IMGSimilar(a.F,c.F,'color=1 shape=1 location=1 texture=1', 20,123)=1and c.id=2

Where *SimJoin(a.F, b.F, ε)* is the similarity-based join of M_1 and M_3.

The above query expression can be computed in two possible plans that are algebraically expressed as:

Plan-1: $(\delta_{IMGSimilar(a.F,b.F,,20)}(simJoin(a.F,b.F,\varepsilon)))$

Plan-2: $(simJoin(\delta_{IMGSimilar(a.F,b.F,,20)}(a.F),b.F,\varepsilon)))$

The query execution is performed for plan-1 and plan-2, where plan-2 is optimal according to Rule-2.

The inner table of the similarity-based join is kept constant for each query execution. Four tables of varying sizes (between 50 and 1000 images) are used as an outer table of the join. In the execution of optimized query plan (plan-2), the similarity-based selection is performed on the outer tables of similarity-based join.

For each execution of the query, an execution time for optimized and non optimized plans is measured. For each join case the query is executed 10 times and an average value is calculated.

As shown in Table-1 there is a significant decrease in execution time of the join operation when the heuristic rule-2 is employed. As the chart shows the cost difference is more significant when the size of the join tables gets larger. This is because when the table size gets larger more number of records will be filtered out by the selection operation and few records will be left for the join operation. As computation of similarity-based join is more expensive compared to that of similarity-based selection the cost reduction is evident.

Table 5-3 result of an Experiment on pushing down similarity-based selection

No. Of Records	Execution time Before rewriting (Milliseconds)	Execution time After rewriting (Milliseconds)
50	342	954
100	665	1938
500	4120	19300
1000	10510	40000

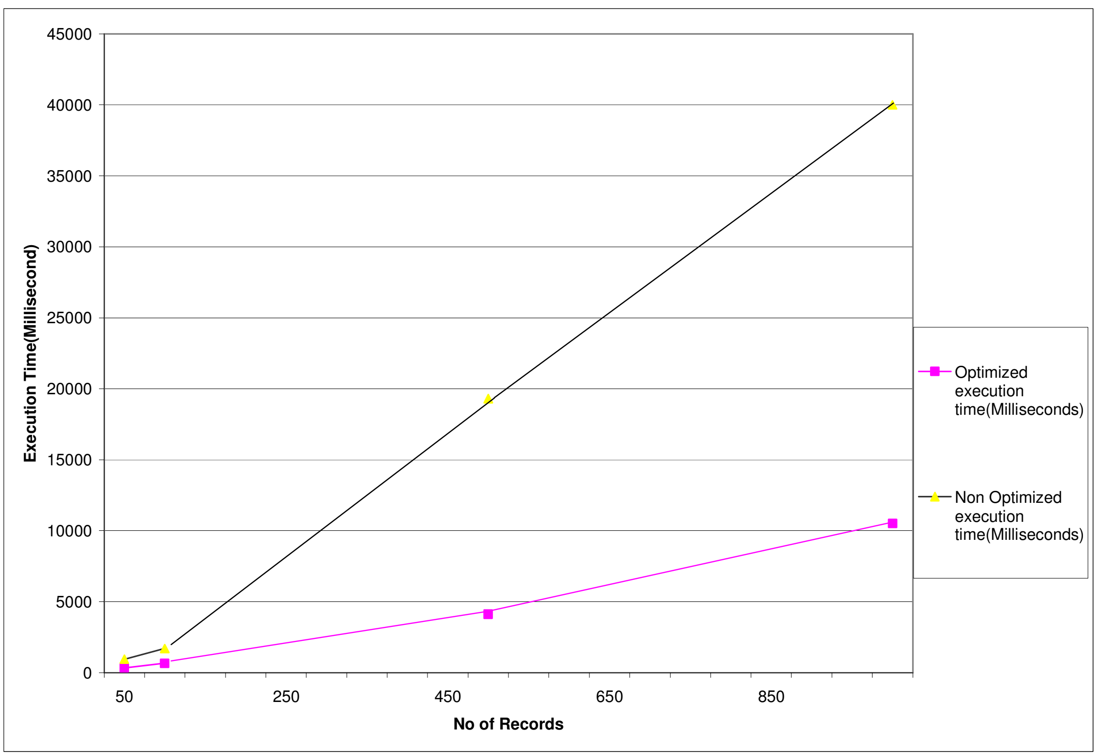


Figure 5-3 Result of Experiment on pushing similarity-based selection before similarity-based join

Experiment-3: Avoid redundant computation of similarity-based selection on multi similarity-based joins.

This experiment is conducted to test performance of Rule-3, the rule of avoiding redundant computation of selection due to push down of selection predicate on multi similarity-based join.

The test is performed on query of a form:

*SELECT a.**
FROM M_1 a, M_2 b , M_3 c ,$M_{4\,d}$, temp e
WHERE (ORDSYS.IMGSimilar(a.F,e.F,'color=1 shape=1 location=1 texture=1', 20)=1 and e.id=2) and (
ORDSYS.IMGSimilar(a.F,b.F,'color=1 shape=1 location=1 texture=1', 20)=1
Add_un
ORDSYS.IMGSimilar(a.F,c.F,'color=1 shape=1 location=1 texture=1', 20)=1
Add_un
ORDSYS.IMGSimilar(a.F,d.F,'color=1 shape=1 location=1 texture=1', 20)=1)

Where *add_un* is additive union operator as defined in chapter 3.

The above query expression can be computed in plan-1 or plan-2 given below. Plan-1 shows the case where selection is pushed to M_1; In this case similarity-based selection on M_1 is computed multiple times. Plan-2 shows the plan the case where redundant computation of similarity-based selection on M_1 is avoided and it is an optimized form according to Rule-4.

The two plans are algebraically expressed as:

Plan-1:
$$\begin{aligned}
&simjoin((\delta_{IMGSimilar(a.F,c.F,,20)}(\mathrm{M}_1)).\mathrm{F}, \mathrm{M}_2.\mathrm{F}, \varepsilon)add_un\\
&simjoin((\delta_{IMGSimilar(a.F,c.F,,20)}(\mathrm{M}_1)).\mathrm{F}, \mathrm{M}_3.\mathrm{F}, \varepsilon)add_un\\
&simjoin((\delta_{IMGSimilar(a.F,c.F,,20)}(\mathrm{M}_1)).\mathrm{F}, \mathrm{M}_2.\mathrm{F}, \varepsilon)
\end{aligned}$$

$$simjoin((\overline{\mathrm{M}_1}).\mathrm{F}, \mathrm{M}_2.\mathrm{F}, \varepsilon) add\_un$$

Plan-2: $$simjoin(((\overline{\mathrm{M}_1}).\mathrm{F}, \mathrm{M}_3.\mathrm{F}, \varepsilon) add\_un$$

$$simjoin((\overline{\mathrm{M}_1}).\mathrm{F}, \mathrm{M}_4.\mathrm{F}, \varepsilon)$$

Where $\overline{\mathrm{M}_1}$ is an intermediate result obtained by evaluating $\delta_{IMGSimilar(a.F,c.F,,20)}(\mathrm{M}_1)$.

The experiment is performed on multi similarity-based join involving four image tables. The size of the inner three tables is kept constant, each with 40 records and that of left most tables is varied between 50 and 1000 as shown in Table-3. For each left most tables the experiment is performed in two cases. One with out applying the optimization heuristic and the second with the optimization rule applied. For each table size the query is executed a number of times and an average execution time is taken. The result shows that there is a significant decrease in execution time. This is because in the optimized case in addition to redundant execution of selection predicate the management of intermediate result produced by the selection predicates is reduced. That is if the selection was to be performed on the left most table for each of the pair wise joins then the intermediate result of each computation has to be written back to disk. As the cost of writing intermediate results back to disk is expensive the cost reduction is evident.

One observation here is that the cost of writing the intermediate result back to disk is expensive and proportional to the size of the intermediate result produced.

Table 5-4 Result of experiment on eliminating redundant evaluation of predicates

No. of Records in M1	No. of Records in M2	No. of Records in M3	No. of Records in M4	Size of intermediate result	After Rewriting (Seconds)	Before Rewriting (Seconds)
50	40	40	40	6	47.94	61.94
100	40	40	40	14	94.53	121.41
500	40	40	40	72	146.86	339.24
1000	40	40	40	144	243.47	624.35

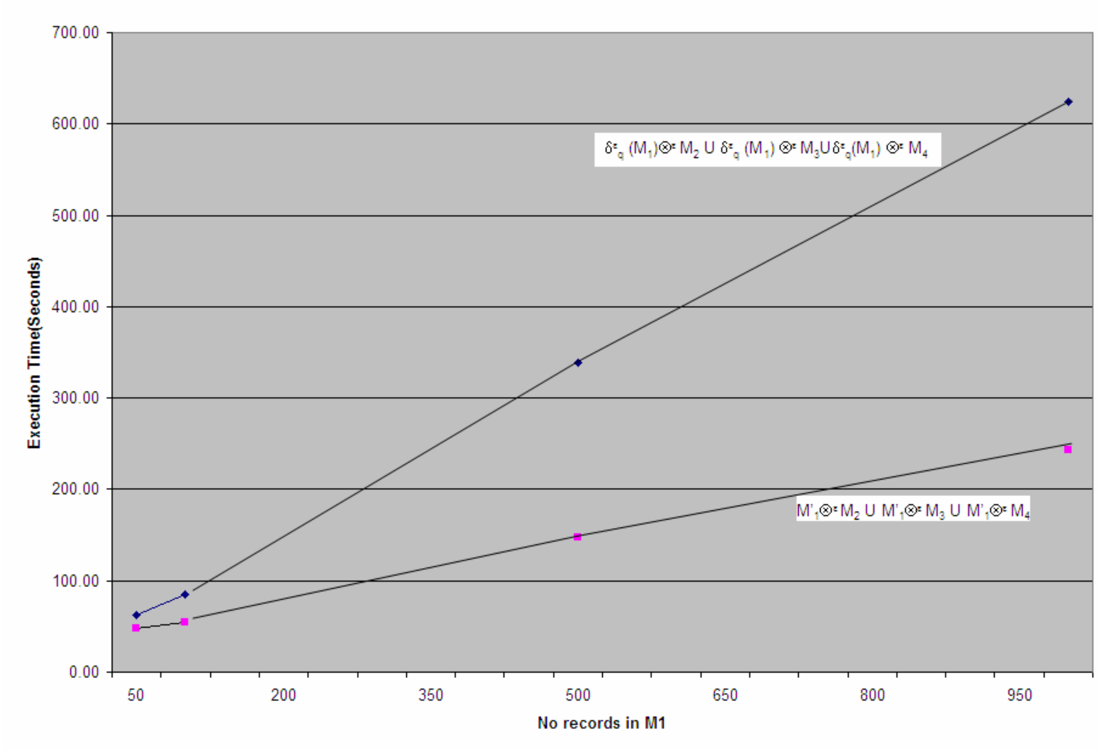


Figure 5-4 Result of experiment on eliminating redundant computation of selection on multi similarity-based join.

Experiment-4: Experimental result for the use of indexing scheme on the F component of the image table

The forth experiment is performed to test the heuristic for use of existing index on the feature vector component of image table. In the environment where we performed the experiment ε-kd-Ttree is used to index image features. The experiment is performed one with the existing index employed to access image features and the other with out employing the index (with full table scan). The experiment is performed on tables with varying number of images (between 50 and 1000).

The test query is of the form:

*SELECT a.**

FROM M_3 a,M_2 b

WHERE ORDSYS.IMGSimilar(a.F,b.F,'color=1 shape=1 location=1 texture=1', 20)=1 and b.id=2 ORDER BY SCORE

The result shows that when index is used to access data from disk there is a much cost saving in terms of execution time. Especially, with a high selectivity and large number of records, the execution time decreases significantly compared to the execution time of the non-optimized query. As shown in the Table-4 and Fig 5-3 the use of existing index on the feature vector component of an image table has highly improved the performance of the query processing.

Table 5-5 Result of Index search on feature vector

No. of Records in a Table	Execution Time without index (Millisecond)	Execution Time with index (millisecond)
50	78	24
100	157	47
500	935	156
1000	1640	297

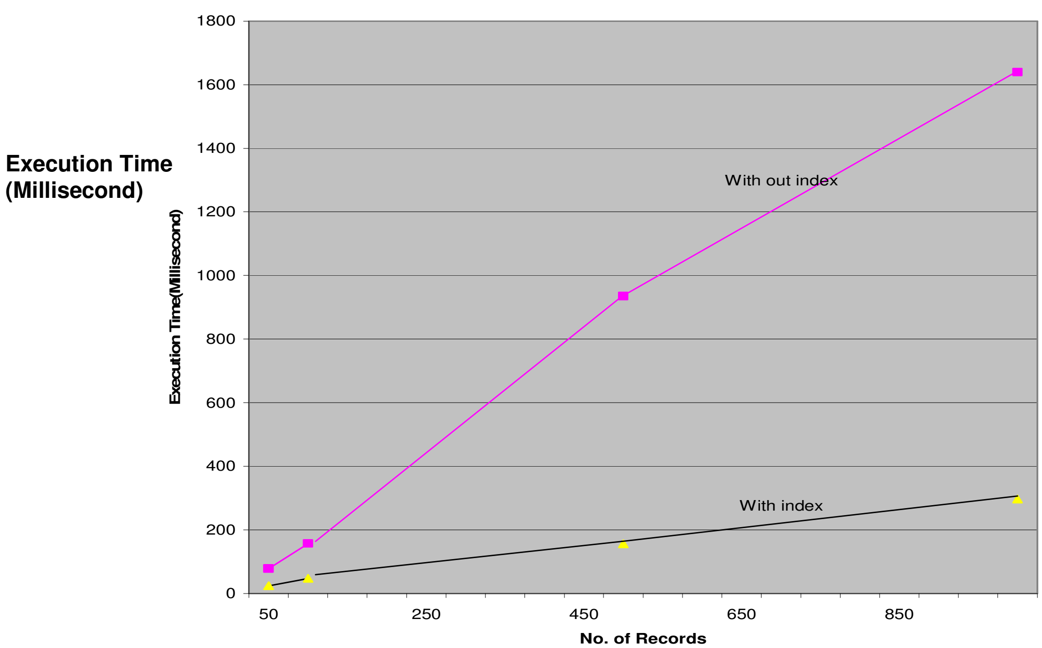


Figure 5-5 Result of Use Index on image feature to access base table.

Chapter 6

Conclusion and Future Works

6.1 Conclusion

In this thesis we have reviewed works in similarity-based image database, similarity-based algebra, and similarity-based query optimization. We have proposed query optimization heuristics for similarity-based image database that use properties of the similarity-based algebra. The proposed heuristic rules are tested to show their performances. The result of the performance test shows that there is a significant decrease in execution time of queries that are optimized using the proposed heuristics.

In addition, we have proposed a method of integrating the proposed query optimization techniques with existing similarity-based image database systems. As the current commercial OR database systems that support similarity-based image query processing do not allow extension of their rule-based optimizer we have proposed to develop the query optimizer and integrate it into the query processing module proposed in previous work [48]. Hence, we have identified the requirements of query optimizer for content-based image database and proposed a design of extensible rule-based query optimizer. We have reviewed the cost model, the search space of query optimizer and the search strategy that can be part of the query optimizer.

The main contributions of our work are that:

- We have developed query optimization heuristics for similarity-based image database
- We have conducted experiments to test the performances of proposed heuristic Rules.

- We investigated and showed the efficiency that can be gained by using multidimensional indexing on feature vector of images.
- We have identified a method of integrating the proposed query optimization techniques with existing similarity-based image database systems.
- We have modeled a query optimizer that meets requirements of query optimization in similarity-based image database.

5.1 Future Works

Future works in the area of query optimization for content-based image database includes:

- Full implementation of the query optimizer,
- Development of efficient Cost model for similarity-based operations of image database and integrate the same with the proposed query optimizer
- Develop more efficient implementation algorithms (physical algebra) for similarity-based join.
- Development and integrate better indexing scheme in image DBMS.
- Extension of Optimization techniques for parallel and distributed database systems.
- Development of query optimization techniques that exploit inter-operator parallelism.

Annex. A Description of class for the query optimizer

<u>Description of the classes for ***CBODBOptimizer***</u>

- The *CBIDBOptimizer* provides an interface for query processor. It defines a method called *Optmize* that receives as an input a query expression and returns an execution plan. *CBIDBOptimizer* implements the Optimizer interface. It coordinates the optimization process. It defines the optimization phases, coordinate rule sets, and search strategies. It delegates the implementation of the search algorithm of the optimization to TopDownSearchStrategy or any other concrete search strategy implemented. It dynamically combines different types of similarity-based algebraic rules and implementation rules, search strategies and cost models.

- *CBIDBSearchStrategy* define an interface for expanding the search space and searching for good expression. It hides one or more specific types of search strategies within an optimizer. *TopDownSearchStrategy* implements a service for query optimization search strategies as Top Down dynamic programming. It collaborates with rules to execute transformation in the query expression expanding the search space and exploiting optimization possibilities. it uses rule factory to create list of rules, Process each rules received from factory against expression, Add the expressions generated by the rules to the search space and determine how to apply the rules to each term in the expression.

- *CBIDBRuleFactory* defines an abstract interface for *LogicalCBIRuleFactory* and *PhysicalCBIRuleFactory*. It represents a rule set which may be defined for a logical algebra or the physical algebra.

- *CBIDBRule* encapsulate a transformation of a term within an expression. It provide mechanism based on pattern matching for verifying if it is applicable to a given term. It provides a method to verify some condition of application, and promise method to give priority to the application of rules. Concrete Rules only four are shown on the diagram implements a specific transformation rule for specific similarity-based algebra operator under some condition. One class is implemented for each of the rewriting rules proposed for the similarity-based algebra and physical algebra. Each class defines a pattern description of the expression it should e applied. It also estimates cost of expression.

Annex. B Cost Formula

1. **Cost formula for Similarity-Based selection**

Based on Euclidean metric method of similarity matching for range query with threshold (ε); the probability of page accesses in low and high dimensional cases are given:

- **Low dimensional data space**

Page access probability for range query in Euclidean metric is given by [35]. The formula estimates the probability with which a give page is accessed by a range query on a uniformly distributed data.

$$X\ (\varepsilon) = \sum \binom{d}{k} \left((1-\frac{1}{C_{eff}}).\sqrt[d]{\frac{C_{eff}}{|M|}} \right)^{k} . \frac{\sqrt{\pi^{d-k}}}{\Gamma\left(\frac{d-k}{2}+1\right)} .\varepsilon^{d-k}$$

The expected number of page access can be obtained by multiplying the access probability with the number of data pages.

$$\mathrm{A}(\varepsilon) = \frac{|\mathrm{M}|}{C_{eff}} .X(\varepsilon)$$

Where |M| is the cardinality of the image table, C_{eff} is the effective capacity of the data page and ε is the radius of the range query and X(ε) is the page access probability.

- **High dimensional data space**

Considering boundary effect for high dimensional case, the probability of page access is given as [35].

$$X_{hd}(\varepsilon) = \sum \binom{d'}{k} \cdot \left(\frac{1}{2} - \frac{1}{4C_{eff}}\right)^{d'-k} \cdot \left(\frac{1}{2} + \frac{1}{4C_{eff}}\right)^{k} V_{csi}\left(k, \frac{r}{\left(\frac{1}{2} + \frac{1}{4C_{eff}}\right)}\right)$$

And the expected umber of page access is:

$$A_{hd}(\varepsilon) = n_{d'} . X_{hd}(\lfloor \log_2 (N/C_{eff}) \rfloor, \varepsilon) + n_{d'-1} . X_{hd}(\lfloor \log_2 (N/C_{eff}) \rfloor, \varepsilon)$$

where

$$n_{d'} = 2.\left(\frac{N}{C_{eff}} - 2^{\log_2 (N/C_{eff})}\right)$$

and

$$n_{d'-1} = \frac{N}{C_{eff}} - n_{d'}$$

Where C_{eff} is the average number of data elements in a node of the R-tree and d is the dimension of the data space and d' is given as $\lfloor \log_2 (N/C_{eff}) \rfloor$

2. <u>Cost formula for Similarity-Based Join</u>

For sequential scan case the cost of joining two tables on similarity predicate is given in [34]. The estimated cost is based on the following assumptions; no cache is used, random order of processing and for each page that must be processed a constant number λ of pages must be loaded from the disk.

$$A = \frac{\lambda |M_1| |M_2|}{\beta^2 / (4d v_{io})^2} \cdot \frac{\beta . t_{tr}}{v_{io}}$$

Where

- d is the dimension, and β is a hardware constant,

- λ is the number of pages loaded from disk for each page to be computed,

- $|M_1| and |M_2|$ are cardinalities of the two tables to be joined and

- v_{io} is the i/o operation overhead and is given by

$$v_{io} = \frac{\beta}{C_{eff}.4d}$$

Where is a hardware constant $\beta = \frac{t_{seek} - t_{lat}}{t_{tr}}$ =40000 for typical hardware.

The computation cost of performing the similarity join when using multidimensional indexing requires estimation of the probability of accessing a page. To estimate join cost assumes independence and uniform distribution of the points in a d-dimensional unit hyper cube $[0, 1]^d$.

The side length data pages is also assumed to be $\sqrt[d]{\frac{C_{eff}}{|M_1|}}$ and $\sqrt[d]{\frac{C_{eff}}{|M_2|}}$ for M1 and M2 respectively.

The index selectivity is computed using Minkiwosky sum.

$$\sigma_{\min} = \sum \binom{d}{i} . \left(\sqrt[d]{\frac{C_{eff}}{M_1}} + \sqrt[d]{\frac{C_{eff}}{M_2}} \right)^{d-i} .V_{i-\dim spahre}(\varepsilon)$$

Where $V_{i-\dim spahre}(\varepsilon) = \frac{\sqrt{\pi^d}}{\Gamma(d/2 + 1)} .\varepsilon^d$ denotes the volume of a hyper sphere with radius ε .

The I/O cost of joining the two image tables based on the similarity predicate is given by

$$A_{Total}(\varepsilon) = \sigma_{\min} . \frac{|M_1|.|M_2|}{C_{eff}}$$

REFERENCES

[1] S.Atnafu, L.Brunie, H.Kosch. ***Similarity-based operators and query optimization for multimedia database systems***, In: Proc. of Int. Database Eng. and App. Symp. (IDEAS), IEEE computer Society, Grenoble, France, july2001, p 346-355.

[2] S.Atnafu,, L.Brunie, L., H.Kosch, ***Similarity-based algebra for multimedia database systems***, in WAIM'2001 Xi'an,china (accepted for publication)july 2001 LNCS.

[3] Soffer and H. Samet, ***Query processing and optimization for pictorial query trees, Visual Information and Information Systems*** - *VISUAL99* (D. P. Huijsmans and A. W. M. Smeulders, Eds.), Lecture Notes in Computer Science 1614, Springer, Berlin, 1999, pp. 60-67.

[4] S. Berchtold, D. A. Keim, and Hans-Peter Kriegel. ***The X-tree : An Index Structure for HighDimensional Data***, Proc. 1996 Intl. Conf. on Very Large Databases,Bombay, India 1996, pps 28-39.

[5] Nepal, S. & M.V. Ramakrishna , ***Query processing issues in image (multimedia) databases***, in Fifteenth International Conference on Data Engineering (ICDE)', March 23-26, Sydney, Australia 1999, pp. 22-29.

[6] M.V.Ramakrishna and S. Nepal: ***Similarity Query Processing in Image Databases***. Technical Report, University Melbourne 2000

[7] M.V.Ramakrishna, and S. Nepal and P.K. Srivastava ***A heuristic for combining Fuzzy results in multimedia databases***. Austerlian Computer Socity Inc. 2001,

[8] S. Adali and P. Bonatti and M.L. Sapino and V. Subramanian, ***A Multi-Similarity Algebra***, in Proc. ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, 402-413.

[9] M. Ortega-Binderberger, K. Chakrabarti. ***Database Support for Multimedia Applications.*** University of Illinois at Urbana-Champaign. S. Mehrota. University of California at Irvine., August 2001.

[10] Y. Rui, T.S. Huang, and S.-F. Chang. ***Image retrieval: Past, Present, and Future***. Journal of Visual Communication and Image representation,10:1-23, 1999

[11] J.Eakins and M.Graham, ***Content-based image retrieval*: **A technical Report to the JISC Technology Application Programme, Institute for image data research, university of Northumbria at Newcastle, 1999.

[12] Cemco C. Veltkamp,Marela Tanase (2002): ***A Content-Based Image retrieval systems: a Survey***. Department of computer science, Utrecht University

[13] Carlo Giovanon S.Pries and Javam C.Machado. ***DORS Database Query Optimizer with Rule Based search Engine***. SugarloafPLoP 2002 Conference.

[14] T. Ozsu and P. Valduriez. ***Principles of Distributed Database Systems***. Prentice Hall, 1991.

[15] M.Steinbrunn, G.Moerkotte, and A.Kemper. ***Heuristic*** *and* ***randomized optimization*** *for the join*. VLDB Journal, 6(3):191--208, 1997.

[16] J.C. Freytag. ***A rule****--****based view*** *of* ***query optimization***. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pages 173--1.180, 1987.

[17] Surajit Chaudhuri(1999) ***Optimization of queries with user defined predicates***. ACM transaction on Database systems Vol 24 June 1999

[18] Joseph M.Hellerstein **,** Jeffrey F. Naughton **. *Predicate Migration: Optimizing Queries With Expensive Predicates*** (with Michael Stonebraker.) *Proc. ACM-SIGMOD International Conference on Management of Data,* Washington, D.C., May,1992

[19] Solomon Atnafu. ***These Modelisation et tratement de requetes images complexes.*** L'Institut National des Sciences Appliquees de Lyon,2003

[20] Vincent Oria, M.Tamer Ozsu,Ling Liu,Xiaobo Li,Jhon Z,Youping Niu and Paul J.Iglinski ***Modeling Images for content-based queries: the DISIMA approach***. Department of Computer science,University of Alberta,1999

[21] C.S.R. PARABHU. ***Object-oriented database systems: Approaches and Architectures***. Prentice Hall of India pvt. Ltd new Delhi-2001

[22] P.Lyman,H.R. Varian***. How much information***?, a research report at a school of Information management and systems at the university of California at Berkely,aRegent of the university of California, October 2000. http://www.sims.berkely.edu/how-much-info/

[23] Selinger PG, Astrahan MM, Chamberlin DD, Lorie RA, Price TG (1979) ***Access path selection in a relational database management system***.In: Proceedings of the

1979 ACM SIGMOD international conference on management of data, Boston, June 1979.

[24] Jarke, M., and Koch, J. ***Query Optimization in Database Systems***, ACM Computing Surveys**,** 16:2, June 1984.

[25] Humid Pirahesh, T. Y. Cliff Leung, Waqar Hasan, ***A Rule Engine for Query transformation in Starburst and IBM DB2 C/S DBMS***, ICDE 1997, pp. 391-400.

[26] Guy M. Lohman, ***Grammar-like Functional Rules for Representing Query Optimization Alternative***, SIGMOD 1988, pp. 18-27.

[27] Hamid Pirahesh, Joseph M. Hellerstein, Waqar Hasan: ***Extensible/Rule-Based Query Rewrite Optimization in Starburst*** *Proc. ACM-SIGMOD International Conference on Management of Data,* San Diego, June, 1992. pp. 39-48

[28] Graefe, G., Volcano – ***An Extensible and Parallel Query Evaluation System***. IEEE Trans. on Knowledge and Data Engineering 6 (1994), 120-135.

[29] Graefe, G., and W.J. McKenna, .***The Volcano Optimizer Generator: Extensibility and Efficient Search.*** Proc. of the 9th Intl. Conf. on Data Engineering, 1993, 209-218.

[30] Stonebreaker, M., ***Object-Relational DBMSs: The Next Great Wave***. Morgan Kaufmann publishers, 1996.

[31] Gail Mitchell,Stanley B. Zdonik,Umeshwar Dayal. ***Object-Oriented Query ptimization: What's the Problem?*** Technical Report No. CS-91-41 June 1991

[32] M. Tamer Ozsu. David D. Straube, ***Queries and query processing in object-oriented database systems***, ACM Transactions on Information Systems (TOIS) (1990), 387–430.

[33] Y. Theodoridis, E. Stefanakis, T. Sellis, ***Cost Models for Join Queries in Spatial Database***, Proceedings of the 14th IEEE Conference on Data Engineering (ICDE), 1998.

[34] Christian Böhm and Hans-Peter Kriegel, ***A Cost Model and Index Architecture for the Similarity Join***, Proc. 17th IEEE Int. Conf. on Data Engineering (ICDE), Heidelberg, Germany, 2001.

[35] Bohm C., ***Acost model for query processing in high dimensional data spaces***, ACM, 2000

[36] Boehm C., Berchtold S., and Keim D.A.: ***Searching in High-dimensional Spaces - Index Structures for Improving the Performance of Multimedia Databases***, ACM Computing Surveys, 2001

[37] Ramez A. Elmasri and Shamkant Navathe, ***Fundamentals of Database Systems, Third Edition***, Addison-Wesley, 2000

[38] Poosla,Ioannidis,Y.Haans p.j, Shekita. ***Improved Histogram for Selectivity estimation of range predicates***. In.Proc. of ACM SIGMOD Montreal.

[39] Surajit Chaudhuri: ***An Overview of Query Optimization in Relational Systems***. PODS 1998: 34-43

[40] Demitrios Gunopulos, George Kollios,Vassillis J.,Carlotrios. ***Selectivity estimators for multidimensional range queries***. The VLDB Journal (2003)

[41] Poosala V, Ioannidis YE, Haas PJ, Shekita EJ (1996) ***Improved histograms for selectivity estimation of range predicates***.In:Proceedings of the 1996 ACM SIGMOD international conference on management of data, Montreal, May 1996

[42] M. Tamer Özsu, Adriana Muñoz, Duane Szafron. ***An Extensible Query Optimizer for an Objectbase Management System (1995)*** Proc. Fourth Int. Conf. on Information and Knowledge Management (CIKM'95)

[43] Poosala V, Ioannidis YE, Haas PJ, Shekita EJ (1996) ***Improved histograms for selectivity stimation of range predicates***.In: Proceedings of the 1996 ACM SIGMOD international conference on management of data, Montreal, May 1996

[44] S. Acharya, V. Poosala, and S. Ramaswamy. ***Selectivity estimation in spatial databases***. In Proc. ACM SIGMOD Int.Conf. on Management of Data, pages 13–24, June 1999.

[45] Muralikrishna M, DeWitt DJ (1988) ***Equi-depth histograms for estimating selectivity factors for multi-dimensional queries***.In:Proceedings of the 1988 ACM SIGMOD international conference on management of data, Chicago, June 1988

[46] MPEG-7 Overview (version 9) international organisation for standardisationorganisation internationale de normalisation iso/iec jtc1/sc29/wg11 coding of moving pictures and audio iso/iec jtc1/sc29/wg11n5525 Pattaya, March 2003 http://www.chiariglione.org/mpeg/standards/mpeg-7/mpeg-7.htm (consulted on 3 May, 2004)

[47] R. Ganski and H. Wong . ***Optimization of nested sql queries revisited***. In Proceedings of the ACM-SIGMOD International Conference on Management of Data, San Francisco, California, pages 23-33, May 1987.

[48] Atnafu S, Chbeir R, Brunie L. ***Efficient Content-Based and Metadata Retrieval in Image Database***. J. USC journal Volume 8, 2002 pp 613-622

[49] Goetz Graefe. ***Query Evaluation Techniques for Large Databases.*** ACM Computing Surveys,Vol. 25, No. 2, June 1993

DECLARATION

I, the undersigned, declare that the thesis is my original work, has not been presented for a degree in any other university, and all sources of material used for the thesis have been duly acknowledged.

Name: **Seifu Geleta**

Signature: ____________

Place: **Faculty of Informatics, Addis Ababa University**

Date of Submission: **July 2004**

This thesis has been submitted for examination with my approval as university advisor.

____

Dr. Solomon Atnafu (PhD)