A PARTIAL PRE-COMPUTATION OF AGGREGATES FOR MOLAP DATABASE

by

Chao Li
B.S., Wuhan University 1997

THESIS
SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

In the School
of
Computing Science

© Chao Li 2003

SIMON FRASER UNIVERSITY

August 2003

All rights reserved. This work may not be
reproduced in whole or in part, by photocopy
or other means, without permission of the author.

APPROVAL

Name: Chao Li
Degree: Master of Science
Title of Thesis: A Partial Pre-computation of Aggregates for MOLAP Database

Examining Committee:

Chair: Dr. Tiko Kameda
Professor

Dr. Woshun Luk
Senior Supervisor
Professor

Dr. Ke Wang
Supervisor
Professor

Dr. Qianping Gu
Examiner
Associate professor

Date Approved: August 6, 2003

SIMON FRASER UNIVERSITY

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project and extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay:

Partial pre-computation of aggregates for MOLAP database

Author: _____
(signature)

Chao Li

_____
(name)

_____ _____
(date)

:vg

ABSTRACT

Partial pre-computation for OLAP (On-Line Analytic Processing) databases has been a popular research topic in recent years. Obviously, a partial pre-computation scheme should be chosen such that the answering time of a particular query workload is optimised. The query workload of an OLAP application is the set of queries users expect. Most of the papers published only deal with the optimisation for the workload of views within the context of ROLAP (Relational OLAP). The partial pre-computation schemes optimised for views have been criticized for lack of support to ad-hoc querying and that a view may be too large a unit for pre-computation for queries that fetch very small answers.

In this thesis, we study the problem of partial pre-computation for the workload of point queries, which is suited for MOLAP (Multidimensional OLAP) systems. The point queries in our workload are range queries defined by the dimension hierarchies. Our study shows that without careful coordination between pre-computation and query processing, it would be difficult to realize any significant gain in query performance from the pre-computation for point queries. We present a new organization of pre-computed cells, Pre-computation Cube or PC Cube, upon which we devise a cover-based querying processing method that is efficient and effective. We also design an algorithm for the selection of the PC Cube. Experiments are performed to show (i) the efficiency of the cover-based query processing method, (ii) the effectiveness of the PC Cube and our selection algorithm, and (iii) the performance comparison of variations of our selection algorithms.

DEDICATION

--- To my father.

ACKNOWLEDGEMENTS

I wish to express my deep gratitude to my supervisor, Dr. Woshun Luk. I thank him for his continuous encouragement, support and guidance with his knowledge and experience. He taught me practices and skills that I will use in my future career. My gratitude also goes to Dr. Ke Wang and Dr. Qianping Gu for time and expertise to improve this thesis.

I would also like to thank the support staff and faculty in our department for always being helpful over the years. Thanks are also due to my friends and great colleagues in Simon Fraser University.

Last, but certainly not least, I would like to thank my family for their unfailing moral encouragement and support. They always show their strong confidences in me. I hope they will be proud of my work, as I am proud of them. Their love accompanies me wherever I go.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Data Warehousing

Since the late seventies, relational database technology has been gaining wide acceptance. Almost every organization relies on it to store, organize, and update their inventories, sales history, customers' information, marketing information, etc., in a collection of large databases. But the fast-growing, tremendous amounts of data collected have far exceeded the human ability for comprehension and examination. As a result, large databases become 'data tombs' and this phenomenon has been described as 'data rich but information poor' problem [FPS96a]. However, in current highly competitive world, the decision makers need to have the right information at the right time. Traditional systems, based on transaction processing, are not well suited for the new requirements. This gives rise to new database technologies, called **Data Warehousing** and **On-line Analytical Processing** (OLAP), and in general, to what is called **Decision Support Systems** (DSS).

Broadly speaking, data warehousing refers to architectures, algorithms, tools and techniques for bringing together data from multiple databases or other information sources, into a single repository suited for querying or analysis [Vas98]. This repository

is called as a *Data Warehouse.* In [Inm92], the data warehouse is defined as a subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision-making process.

1.2 On-line Analytical Processing

Among many available technologies of analysing data in a data warehouse, On-line Analytical Processing (OLAP) is one of the most popular ones for on-line, fast, and effective multidimensional data analysis [CD97].

OLAP is a category of software technologies that enable analysts, managers and executives to gain insights into data through fast, consistent, interactive access to a wide variety of possible views of information transformed from raw data to reflect the real dimensionality of the enterprise as understood by the user [OLAP97]. OLAP, as an interactive decision-support process, has been used extensively in both database and data warehouse applications, and enjoys continuous increase of market share in recent years.

1.3 Multidimensional Model

Current OLAP applications are based on a multidimensional data model that employs multidimensional arrays for modelling data [CD97]. Data are considered as points in a multidimensional array (also called as Hypercube, or Cube). A cube is defined in [OLAP97] as a group of data cells arranged by the dimensions of the data. Each cell is uniquely defined by the corresponding values of the dimensions of the cube. The content of the cell is named *measure*, which is of users' interest.

The following example considers a database that consists of records of sales information for products sold in some city at some time. There are three dimensions: *product*, *Location*, and *time*. The measure of interest is the *sales*. The three-dimensional array below can be viewed as a cube with each dimension forming a side of the cube ([Figure 1.1]).

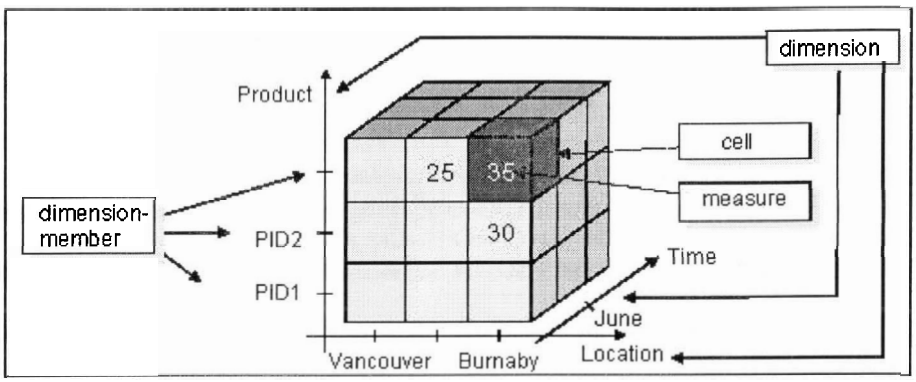


Figure 1.1 A multidimensional cube

A *Dimension* is a structural attribute of a cube, which is a list of *members* that are of a similar type in the user's perception of the data [OLAP97]. A dimension acts as an index for identifying values within a multi-dimensional array. If one member of a dimension is selected, the remaining dimensions in which ranges of members (or all members) are selected define a *sub-cube*. If all dimensions have a single member selected, then a single cell is defined.

A *dimension* shows all possible ways in which the user can meaningfully group the detailed information stored in the multidimensional database. Each dimension is organized in a *hierarchy* of *levels*, corresponding to data domains at different granularities. Each level is associated with a set of members. Among the different levels of a dimension hierarchy, the highest level is the level *All*, so that we can group all the

values of the dimension into the single value '*All*'. The lowest level is called the *detailed level* of the dimension. The members in the lowest level are called as *primary members*, and all other members in the higher levels of the dimension hierarchy are called as *group members* [Luk01]. The following graph (Figure 1.2) is an example hierarchy of dimension *Location*. Province *BC* is a group member; s_1 and s_2 are primary members.

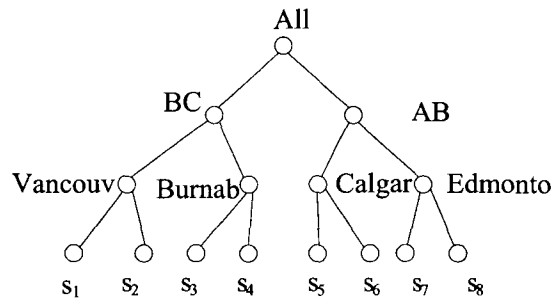


Figure 1.2. Hierarchy of dimension *Location*

1.4 Implementation Approaches for OLAP

According to the actual physical storage of data, there are two main implementation approaches for OLAP systems: **Relational OLAP** (ROLAP) and **Multidimensional OLAP** (MOLAP).

ROLAP is based on a relational database server, extended with capabilities such as extended aggregation and partitioning of data [CD97]. The aggregates and metadata are stored in tables. An extra module or application is needed to translate the tables into cubes for multidimensional data analysis. The schema of the database can be a star, snowflake, or fact constellation schema [CD97].

MOLAP is based on "pure" Multidimensional Database (MDB), which logically stores data in multidimensional arrays. The multidimensional arrays are heavily

compressed and indexed on the physical level for space and performance reasons [Vas98].

The following table (Table 1.1) shows some important differences between ROLAP and MOLAP architectures [Vai98].

	ROLAP	MOLAP
Storage and Access	- Tables/tupels - SQL access language - Third party tools	- Proprietary arrays - Lack of a standard language - Sparse data compression
Usage	- Variable performance - Relational engine	- Good performance - Multidimensional engine
Database size	- Gigabyte -Terabyte - Large space for indexes - Easy updating	- Gigabyte - 2% index space - Difficult updating

Table 1.1 ROLAP Vs MOLAP

ROLAP systems usually bring a larger storing capacity and are more scalable than MOLAP systems. However, MOLAP systems tend to offer better performance than ROLAP systems. Actually, there was a long debate in the database community about which approach should be used. However, the debate seems to be settled at present because few pure ROLAP systems survive and many major ROLAP vendors have recently incorporated MOLAP facility into their products, like the Oracle 9i and MicroStrategy 7i.

1.5 OLAP Query Performance

An OLAP query usually involves an aggregation over a large amount of data in database, which may take a long time to complete. Therefore, improving the response time of OLAP queries is of the interests to many researchers, and many techniques have been developed. Those techniques can be categorized into two classes: the techniques using pre-computation and those without pre-computation. The techniques that do not involve pre-computation mainly use some special data structures or physical storage methods [DRT99], which may allow fast access to the primary database and provide support to ad hoc querying. These techniques are only suitable for the databases in modest size. As a result, many researchers adopt the pre-computation as the approach to improve the OLAP query performance.

1.5.1 CUBE BY Operator

The CUBE BY operator is introduced in [GBLP96] for conveniently supporting the computation of multiple group-bys in OLAP. The computation of CUBE BY is useful for answering the OLAP queries that use aggregation on different combinations of dimension attributes. The CUBE BY operation computes the group-bys corresponding to all possible combinations of a list of attributes. That is, a CUBE BY on n attributes corresponds to 2^n group-bys.

For the example database with 3 attributes: *Product*(*P*), *Location*(*L*) and *Time*(*T*), the collection of aggregate queries can be expressed using the CUBE BY operator as the following:

```
SELECT P, L, T, SUM(sales)
FROM R
CUBE BY P, L, T;
```

The CUBE BY operation will result in the computation of 8 possible group-bys: *PLT, PL, PT, LT, P, L, T, ALL*, where *ALL* denotes the empty group-by. Each group-by is often called as a *view* or *cuboid.* The views can be store as tables in ROLAP systems or arrays in MOLAP systems.

1.5.2 Efficient Computation of CUBE BY

Obviously, one can use a brute-force approach to compute each view in the CUBE BY independently. The performance of this approach is very poor. Therefore, many researchers study the efficient computation of CUBE BY, i.e., how to compute 2^n views for a relation with n dimension attributes. Some more efficient pre-computation algorithms have been presented in [AAD+96, SS99, YL99] by using a number of optimisations, such as computing a view from another previously computed view and overlapping the computation of several different views. In [ZDN97, LRS99], algorithms for the pre-computation of aggregates in MOLAP have proposed, where aggregates are stored in one array or multiple arrays. Another efficient computation algorithm in MOLAP is introduced in [Luk01], which considers the relationships among the individual aggregates and stores all the aggregates in one multidimensional array that is compressed into a single linear array in physical level.

However, the pre-computation of all views in CUBE BY usually is not feasible because it often exceeds the available storage limit and incurs a high maintenance cost. Pre-computing all views has also been thought as not the most cost beneficial approach. It has been pointed out in [SDN98] that the gains from more pre-computation outweigh the cost of additional disk space after some degree of pre-computation. So, a practical

alternative is partial pre-computation of OLAP databases, which is also the subject matter of this thesis.

1.6 Partial Pre-computation

The objective of a partial pre-computation strategy is to select a certain amount of aggregates to compute before querying time, so that the query answering time is optimised. Two major issues about this optimisation process are: (i) How many aggregates should be computed? (ii) What kind of queries that a pre-computation strategy is optimised for? The first question depends very much on the storage available. Microsoft, as the vendor of one of the popular OLAP systems, also suggests that 20% of all possible aggregates should be computed.

The most obvious answer to the second question is that these queries should be those that users of an OLAP application expect. This answer though is not easy to characterize because it varies from one application to another, sometimes, from one user to another. In order to study this important question systematically, one needs to precisely identify the set of queries as an object for optimisation. In this thesis, we call this set of queries that the users expect, the query workload[1].

[1] The workload is characterized by the user's explorative and navigational data analysis task and shows specific high-level patterns that stem from the structure of the analytical task the user is solving [SAP99].

1.6.1 Views as Workload

Most researchers have chosen the set of all possible views as the query workload, based on which their partial pre-computation strategies are devised. The workload in [HCKL00] is a given set of views. In [BPT97], all possible views with varying weights are considered. The workload of all possible views with identical weights is studied in [HRU96, GHRU97, SDN98]. In those papers, different schemes are proposed to select a set of views for materialization under some constraints, which are optimised to answer the view queries. A constraint can be a limitation of storage space or a restriction on maintenance time. They assume that the cost of answering a query is related to the sizes of views from which the answer of the query can be computed, even for the queries that may ask for the single cells.

Partial pre-computation schemes that are optimised for views have been criticized for lack of support for ad-hoc querying ([BK99, DRT99, KR99]). The optimizations for the workload of views may not be suitable for ad-hoc querying. It has also been mentioned in [KR99] that many OLAP queries may fetch very small answers, for example, the total sales of a product sold in last ten years. With workload expressed in terms of views, their proposed solutions cannot be guaranteed to always work well for all possible queries, including non-view ones.

1.7 Thesis Objective

In thesis, we study the same optimisation problem, how to select the aggregates for pre-computation such that the query answering time is optimised, but with a different query workload. We will first describe what this workload is and then justify why we adopt this query workload.

1.7.1 Extended Multidimensional Space and Point Queries

In the multidimensional model, data are considered as points in a multidimensional array. For example, tuples in the primary database in an OLAP system can be thought of as points in a multidimensional space. This is a cube, each side of which consists of all primary members of that dimension. The primary members are the members in the lowest level of a dimension's hierarchy and are values that have been exposed in the primary database. Moreover, the tuples in the primary database and the aggregates can also be modelled into points in one single multidimensional space, called as *extended multidimensional space* in [Luk01]. This can be seen as a cube, each side of which consists of the primary members and all group members. Group members are those in the higher levels of a dimension's hierarchy. In the extended multidimensional space, the cells that correspond to the tuples in primary database are called as primary cells, and other cells as aggregation cells.

A point query is to ask for the content of a cell in the extended multidimensional space. More specifically, we are interested in the point queries on the aggregation cells. Actually, the point queries on aggregation cells are range queries over the primary cells.

Let's consider an example database with 3 attributes: *Product*(*P*), *Location*(*L*) and *Time*(*T*). A point query can be:

- *Q1* (*P1*, *BC*, *2002*): What is the total sale of product *P1* sold in province *BC* in *2002?*

  P1 is a primary member in dimension *Product*. Province *BC* and *2002* are group members in dimension *Location* and *Time*. This query asks for the content of an aggregation cell, (*P1*, *BC*, *2002*). This query requires an aggregation over the cells that contain the sales of *P1* sold in all cites in province *BC* from January of *2002* to December of *2002*

- Q2 (*P1*, *Canada*, *2002*): What is the total sale of product P1 sold in Canada in 2002?

  P1 is a primary member in dimension *Product*, and *Canada* and *2002* are group members in dimension *Location* and *Time*. This query asks for the content of an aggregation cell, (*P1*, *All*, *2002*), which is a summation of the sales of *P1* sold in all cites in *Canada* from January of *2002* to December of *2002*.

If the content of an aggregation cell that a point query asks for has been pre-computed, we can directly return the answer to user. If not, the answer of a point query can be computed by accessing the primary cells and/or the pre-computed aggregation cells.

Obviously, the pre-computation of some point queries may help the answering of other point queries. In the above example, if the answer of point query *Q1* has been pre-computed and used to answer the point query *Q2*, then we do not need to access the primary cells which contains the sales of *P1* sold in all cites in province *BC* from January *2002* to December of *2002*.

1.7.2 Justification for Workload of Point Queries

We define our query workload as the set of point queries that ask for the contents of the aggregation cells in an extended multidimensional space. The point queries in our workload are actually range queries over the primary cells, whose ranges are pre-defined by the natural dimension hierarchies, like the example in section 1.7.1. It is more rational to consider the range queries whose ranges are defined according to the dimension hierarchies than the queries with arbitrary ranges. For many practical situations, arbitrary ranges are not really meaningful to users. For instance, grouping

products by a range of item numbers does not make much sense. Users usually categorize the products in a way that is meaningful to them; this categorization is often registered in the dimension hierarchy.

As pointed out in [KR99], many OLAP queries only ask for the contents of few aggregation cells, like, the total sales of a product sold in last 10 years. The optimization for the workload of point queries is well suited for those OLAP queries that require small answers. To the best of our knowledge, there is no published paper about the optimization for point queries. Moreover, many OLAP queries are ad hoc queries [SWS02], which may require computation on demand. Those ad hoc queries may be decomposed into a set of point queries in the extended multidimensional space or require the answer of a set of point queries for computation. The fast computation and processing of the point queries may help to answer those ad hoc queries.

The workload of point queries can be easily fitted into a MOLAP architecture, which stores data in multidimensional arrays and is credited for fast access to individual records in the database. In MOLAP systems, queries are modelled as points or sub-cubes with ranges in several dimensions [Goil99], which can most likely be seen as a set of point queries. The queries are processed and answered by accessing the points, not ranges, in a multidimensional array because cells in a multidimensional array can be only sequentially stored in one combination of attributes at the level of physical storage. Our optimisation scheme for point queries can be easily adapted to MOLAP systems.

1.7.3 Our Approach

There are some special concerns about the point queries. First, there are numerous point queries in our workload. It is impractical to propose a partial

pre-computation scheme which evaluates every single point query and then decides a set of point queries for pre-computation.

There are two types of cost associated with the answering of a query, the *processing overhead* and the *cost of accessing*. The processing overhead is defined to be the cost of searching for relevant pre-computed cells. The cost of accessing is related to how many pre-computed cells that should be accessed in order to compute the answer of a given query. Most optimisation schemes for the view queries ignore the processing overhead. But for the answering of the point queries, the processing overhead is an important consideration because point queries are much more numerous and the answers of point queries are usually smaller. It is unacceptable to spend a lot of time in just locating cells for accessing. So, when we design an optimisation scheme for point queries, we will consider devising a strategy that is low in the processing overhead and cost of accessing.

1.8 Thesis Organization

In Chapter 2, we review some of the related works in OLAP. In Chapter 3, we first demonstrate the challenges associated with processing point queries, and then show how we meet these challenges by introducing a cover-based query processing method. This method requires all pre-computed cells form one cube, Pre-computation Cube or PC Cube. In Chapter 4, we describe a method for the selection of PC Cube. In Chapter 5, we present the experimental results that show the overhead of the query processing and the comparisons of the effectiveness of the member selection method. Chapter 6 summarizes this study and discusses the future research issues.

CHAPTER 2

RELATED WORKS

In this chapter, we introduce the view lattice that depicts the relationships existing between views. Then we briefly review the algorithms on the pre-computation of views, view selection and answering query using the materialized views.

2.1 The View Lattice Framework

The CUBE BY in [GBLP96] will result in the computation of views, which correspond to the SQL queries grouping on all possible combinations of the dimension attributes. Those views are usually denoted by the grouping attributes, e.g.: *PLT, PL, PT, LT, P, L, T,* and *All* for the example database with three attributes *Product*(*P*), *Location*(*L*) and *Time*(*T*), in Section 1.5.

A lattice is used in [HRU96] and [BPT97] to depict the relationships that exists between views. Each node in the lattice represents an aggregate view (can also be called as cuboid or group-by). An edge exists in the lattice from node *i* to node *j*

whenever view *j* can be computed from view *i* and view *j* contains exactly one attribute less than view *i*. In this case, view *i* is called a parent of view *j*. In the lattice, there is a basic view upon which every view is dependent. There is a complete aggregation view '*ALL*', which can be computed from any other view in the lattice

The Figure 2.1 shows the lattice of views for the example database with three dimensions, *Product*, *Location*, and *Time*, which are represented by *P*, *L* and *T* respectively. A view is labelled by the name of the dimension it is aggregated on. In Figure 2.1, view *PLT* is the basic view and a parent of view *PL*.

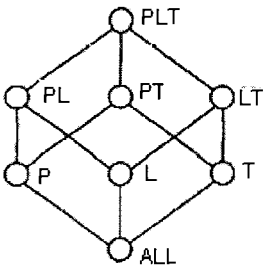


Figure 2.1 A view lattice

2.2 Pre-computation of Aggregates

OLAP queries usually involve a lot of aggregation on a large amount of data in data warehouses. The pre-computation of aggregates can greatly improve query performance. By utilizing the computation dependence among views, many researchers develop their pre-computation algorithms to efficiently compute all possible views, which is so called view materialization.

In [AAD+96, DANR96], several efficient view materialization algorithms have been presented, e.g., Pipesoft, PipeHash and Overlap, which incorporate several optimisation techniques, such as computing a view from its smallest previously computed parent, using the data sorting in a particular order to compute all views that

are prefixes in that order, and computing the views with common prefixes in a pipelined fashion. In [YL99], there are some efforts to study how the skewed data may affect the pre-computation of aggregates and an approach to dynamically manage the memory usage is proposed.

A comparison is shown in [ZDN97] about the difference between the view materialization in ROLAP and MOLAP, and an array-based pre-computation algorithm for MOLAP is proposed. This algorithm stores the partitions of views in main memory arrays, and overlaps the computation of different views while using minimal memory for each view. In [LRS99], the pre-computation on compressed MOLAP database is examined and some algorithms for the computation of views without decompression are suggested.

In [Luk01], another pre-computation algorithm for MOLAP is presented. One distinct feature of the ADODA in [Luk01] is that the aggregation cells are treated in the same way as the source data (primary cells). Both the aggregation cells and the primary cells are stored together in one single data structure, one multidimensional array, which allows them to be quickly accessed. The pre-computation in ADODA considers the points in multidimensional space. ADODA examines the coordinates of the cells and relies on the relationships among cells to determine how to perform aggregation. A graph-theoretic model is employed to ensure the correctness of the summation computation. The complexity of ADODA has been so gracefully managed that only a single scan of the database is required.

2.3 *View Selection*

A full pre-computation, which pre-computes all possible aggregates, can provide the best query performance, but usually it is not advisable for the following reasons:

- A full pre-computation usually requires a great amount of space to store the aggregates. It often exceeds the amount of space available.
- A full pre-computation is not the most cost-beneficial approach. In [SDN98], it has been mentioned that the gains from more pre-computation outweigh the cost of additional disk space after some degree of pre-computation.
- A full pre-computation usually incurs a high maintenance cost.

Because of these reasons, many researchers study the problem of partial pre-computation, which is to choose a set of views for materialization. In [HRU96], an approach is proposed on how to choose a set of views for materialization under a limited storage space. They introduce a linear cost model, which assumes that the cost of answering a query is related to the size of view from which the answer of the query can be computed. The linear cost model has been verified by the experiments. The greedy view selection algorithm in [HRU96] tries to decide which aggregate views to be pre-computed in order to minimize the query cost. The greedy view selection algorithm first chooses the base view, which is the root in the view lattice. Materializing one more view may allow some queries be answered by a smaller materialized view, so that the query cost can be reduced. Thus, their algorithm chooses the view that produces the largest reduction in query cost. The greedy algorithm repeats this selection process and terminates when the total size of the selected views exceeds the space limit. In [HRU96], it has been proved that the benefit of the aggregate views selected by greedy algorithm is no worse than $(0.63 - f)$ times the benefit of an optimal selection, if no aggregate view

occupies more than some fraction *f* of the total space available for pre-computation. The time complexity of greedy algorithm is $O(k * n^2)$, where *k* is the number of views selected and *n* is the number views in the lattice.

In [SDN98], another view selection algorithm, PBS (Pick by Size), is proposed to address the same problem. The difference is that PBS selects the views solely based on the size of the views. In each round, the view with the smallest size among the unselected views is chosen until the total size of the selected views reaches the space limit. PBS is much simpler and faster than the greedy algorithm in [HRU96]. It has been stated in [SDN98] that PBS is so fast that it will enable database administrators to determine the points, at which diminishing returns outweigh the cost of additional storage space, and how much space should be allocated for pre-computation. It has been proven in [SDN98] that the PBS has the same performance as the greedy algorithm in terms of the optimal benefit for a subclass of view/cube lattice called SizeRestricted (SR) Hypercube.

2.4 Query Processing

For efficient processing of OLAP queries, a commonly used approach is to store the results of frequently issued queries in summary tables, i.e., materialized views (or cubes), and makes use of them to evaluate other queries. This approach involves rewriting queries using materialized views (cubes).

In [CKPS95], the traditional query optimisation algorithms are generalized to optimise the query in the presence of materialized views. Some techniques of rewriting a given SQL query are proposed in [SDJL96] such that it uses one or more materialized

view. They also suggest a semantic approach to determine whether the information existing in a view is sufficient to answer a query.

An efficient query rewriting method is suggested in [PKL02], which can utilize the materialized views having different granularities, selection regions and aggregation granularities. In general, for a given OLAP query, there can be many equivalent rewritings using different materialized views/cubes in various ways. Their execution costs are different from one another. An algorithm is also presented in [PKL02] to determine the set of materialized views used in query rewriting.

CHAPTER 3

COVER-BASED QUERY PROCESSING

In this chapter, we first explain some multidimensional terminology that will be used frequently in the rest of this thesis. Then, we discuss how to answer a point query when the contents of a certain amount of points in the extended multidimensional space have been pre-computed. We show that without careful coordination between pre-computation and query processing, it is difficult to realize any significant gain in query performance from the pre-computation. As a result, we propose a query processing method that will make effective use of the pre-computed points provided that all pre-computed points form a cube. Finally, we discuss how we devise the strategy for efficiently processing the point queries.

3.1 Multidimensional Terminology

3.1.1 Dimension Hierarchy

Data are considered as points in a multidimensional cube. Each dimension of the data cube is organized as a *hierarchy of levels* corresponding to data domains at

different granularities. Each *level* is associated with a set of *members*. Based on this, we define the following terminology:

- *primary member* : members at the lowest level of the dimension hierarchy. Primary members essentially are the values that are exposed in the primary database.
- *group member* : all other members in a higher level of the dimension hierarchy. A group member is an aggregation of some other members (primary members or group members)

There exists a child-parent relationship between the members in the dimension hierarchy.

- *child member* of a group member, *m* : a member that is aggregated into the group member *m* and is at one level below *m*.
- *descendent member* of a group member, *m* : a child member of *m*, or a child of a descendent member of *m*.
- *parent member* of a member, *m* : the member which *m* should be aggregated into and is one level above *m*. ***In this thesis, we assume that a member can only have one parent member.***
- *ancestor member of a member, m :* parent member of m, or parent of an ancestor member of *m*.

For the example dimension hierarchy in the following figure (Figure 3.1), $\{a_1, a_2, a_3, a_4, a_5, a_6\}$ is a set of descendent members of a_7. $\{a_5, a_6\}$ is a set of children members of a_7.

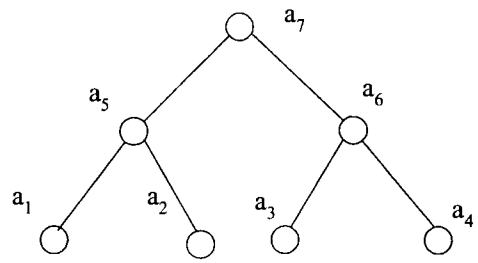


Figure 3.1 An example dimension hierarchy

3.1.2 Cell and Cell Addressing

As mentioned in section 1.7.1, both the aggregates and the tuples in the primary database can be considered as cells in the extended multidimensional space. A cell is uniquely defined by its corresponding members of the dimensions of the extended multidimensional space. The address of a cell T is $(m_1, m_2, ..., m_k)$, where k is the number of dimensions and m_i is the corresponding member in the *i-th* dimension. The address of a cell can also be called as the coordinate of a cell. We can define the primary cell and aggregation cell as the following:

- *primary cell* (*tuple*): the cell whose coordinate contains only the primary members.
- *aggregation cell* (*tuple*): the cell whose coordinate contains at least one group member

Aggregation cells and primary cells are related to each other on the basis of child-parent relationship among the members in the same dimension hierarchy. A cell T_1 is said to be a *child cell* of another cell T_2 restricted to dimension i if the coordinate of T_1 is identical to the coordinate of T_2 except that the *i-th* member of T_1 is a child member of the i-*th* member of T_2. The *descendent cell*, *parent cell* and *ancestor cell* can also be defined in the same way.

3.1.3 Point Query

In chapter 1, we introduced the extended multidimensional space and point query. In this section, we present the formal definition of the point query.

Definition 3.1. **Point query** is defined as: $Q(m_1, m_2, \ldots, m_k)$, which asks for the content of a cell at $(m_1, m_2, \ldots, m_k)$ in the extended multidimensional space, where m_i is a member in dimension i, and k is the number of the dimensions in the relation.

A point query will return the measure of a cell in the extended multidimensional space. If any of the members in the coordinate of the point query Q is a group member, the answer of Q will be the measure of an aggregation cell, which is an aggregation over a set of primary cells.

Definition 3.2. The **processing overhead** of a query is the time spent in determining the set of cells that should be accessed in order to compute the answer of a given query.

Definition 3.3. The **accessing cost** of a query is the time spent in accessing a set of cells required and is defined to be the number of cells accessed in computing the answer of a given query.

Definition 3.4. The **cost** of answering a query is defined to be the sum of processing overhead and accessing cost of a given query.

In next few sections, we will study how to use the pre-computed cells to answer the point query in the partial pre-computation of OLAP database. Our focus is on how to devise a query processing strategy with low processing overhead.

3.2 Process of the Point Query

3.2.1 Query Processing --- No Pre-computation

The answer of a point query is the measure of the cell that corresponds to the query. We define the measure of a cell T to be a non-holistic aggregate function ([GBLP96]) of all measures of cells that are collectively represented by T. The measure is recursively defined:

1) If the children of T are primary cells, the measure of T is a function of measures of all these children.

2) If the children of T are aggregation cells, the measure of T is a function of measures of all its children of T along any dimension i.

The desired measure can be computed via the above recursive method in a top down manner. Alternatively, the result that follows from the above definition, and is proven in [Luk01], offers another way to compute the measure of a cell.

Lemma 1: The measure of T is an aggregate function of measures of all primary cells that are also descendent of T.

Lemma 1 shows that the measure of an aggregation cell can be computed from the set of all its primary descendent cells.

The above discussion suggests two methods to compute the answer of a point query in case of no pre-computation. The top down approach used in [Luk01] is probably the one that most would choose. Starting from a given query Q, the top down approach locates the children of Q along one dimension, and so on, in a way that follows the definition of the measure of a cell. Alternatively, we can follow a "bottom up" approach

according to Lemma 1, which is to compute the answer of Q from the set of all primary descendent cells of Q.

3.2.2 Query Processing and Partial Pre-computation

The strategies in section 3.2.1 work fine in case of no pre-computation. But in partial pre-computation, neither of the strategies would work out if we do not carefully choose what to pre-compute. Let us, for example, take a greedy approach to select the cells for pre-computation, which is similar to the greedy algorithm in [HRU96] for view selection. After the pre-computation phase, a set of cells, say S, has been pre-computed. Now we will try to answer a point query $Q(m_1, \ldots, m_n)$ by using the pre-computed aggregation cells. We definitely do not use the bottom-up method, because it will bypass the set of pre-computed aggregation cells entirely. The top down method, on the other hand, has to deal with two issues associated with query decomposition: how to efficiently determine the child-parent relationships between the query and the pre-computed aggregates, and how to find a good decomposition.

In the top down approach, the query Q will be first decomposed into its children. For each child of Q, the set of pre-computed cells S will be searched to check whether the measure of each child of Q is available. If some of the children of Q have not been pre-computed, they have to be decomposed and processed against S repeatedly. This can be a fairly expensive process.

Assuming now that we somehow are able to determine all child-parent relationships of cells efficiently, let us consider how a query may be decomposed against the pre-computed cells. Figure 3.2 shows an example of the dimension hierarchies of a 2-dimensional database. The dimension attributes are A and B. The members that are

numbered within the interval [1,4] are primary members, and the remaining ones are group members.

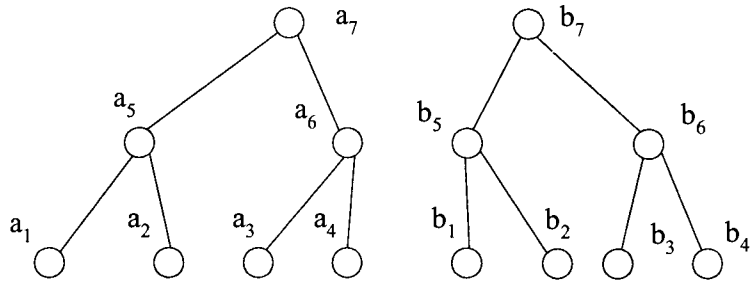


Figure 3.2 Dimension hierarchies for dimension *A* and *B*

The following figure (Figure 3.3) shows how every cell (or point query) in the OLAP database can be decomposed.

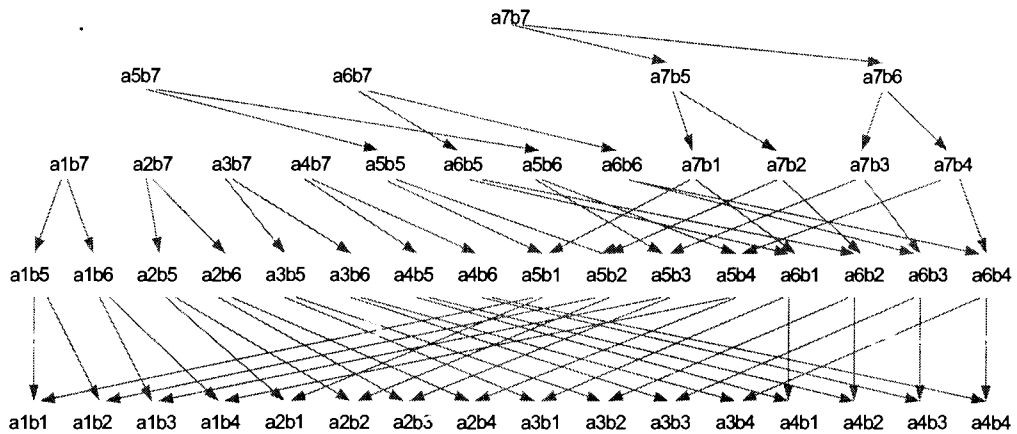


Figure 3.3 Query decomposition – All queries

We will use the above figure to show that it is not easy for top-down query decomposition method to choose a good decomposition. Suppose that the query *Q* is (a_7, b_7) and there are two pre-computed aggregation cells, (a_5, b_7) and (a_6, b_7). The above graph (Figure 3.3), which shows one possible decomposition of each cell, cannot utilize the pre-computed cells to help to answer the query *Q*. The decomposition of *Q* in Figure 3.3 requires 4 rounds of decompositions and accessing total 30 cells. Indeed, the

query $Q(a_7, b_7)$ can be decomposed into (a_5, b_7) and (a_6, b_7), which is obviously a much better choice. Unfortunately, this decomposition could not be chosen unless we are aware of these child-parent relationships before the top-down decomposition begins.

To conclude, the top down method is not a good choice in case of partial pre-computation. We have to resort to some version of bottom-up method. But straightforward bottom-up method, which accesses all primary descendent cells of the query, would not help as it would bypass all pre-computed aggregates. One alternative approach is to carefully choose the cells for pre-computation such that it will make the query processing efficient. In the remaining of this chapter, we will present a query processing strategy, assuming that the set of pre-computed cells satisfy certain conditions.

3.2.3 Query Processing in 1-Dimensional Database with Partial Pre-computation

In order to gain an appreciation of how the point query can be efficiently processed and answered by using the pre-computed cells, we will start with a very simple OLAP database, e.g., a 1-dimensinoal database. In the dimension's hierarchy, we define:

Definition 3.5. Primary Descendent Members of a member m, $pdm(m)$, is the set of primary members that are also descendents of m.

In a 1-dimensional database, the primary descendent members of a member m can be thought as the primary descendent cells of m. According to Lemma 1, the measure of m is computable from its primary descendent members.

Definition 3.6. Cover of a member m, $\{m_1, \ldots, m_n\}$, is a set of members if $\bigcup_{i=1}^{n} pdm(m_i) = pdm(m)$ and $pdm(m_i) \cap pdm(m_j) = \varnothing$ for $1 \le i \ne j \le n$.

The following lemma, which is a strict forwardness of non-holistic function in [GBLP96], will be stated without proof and shows the relationship between a member and its cover.

Lemma 2: The measure of m, which is the result of a non-holistic aggregate function over $pdm(m)$, is computable from the cover of m.

Let's consider an example of 1-dimensional database, whose dimension hierarchy has 3 levels (as in Figure 3.4). Each member in the dimension hierarchy is also a cell in the database, a 1-dimension array (as in figure 3.5).

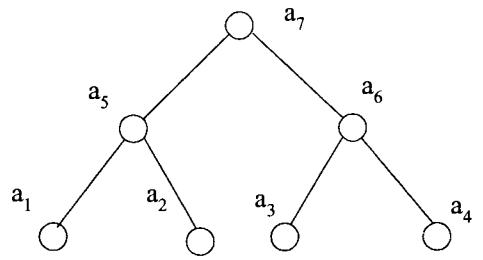


Figure 3.4 Dimension Hierarchy

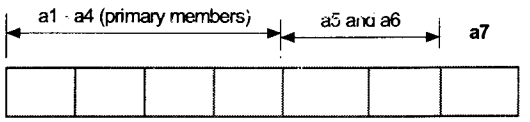


Figure 3.5 Extended 1-dimensinal space

Let $Q(m)$ be a query on this database and m be a_7, the root of the hierarchy that has not been computed. Query $Q(a_7)$ can be computed from $pdm(a_7)$, one of a_7's covers. This will require accessing 4 cells, $\{a_1, a_2, a_3, a_4\}$. Now, suppose another cover of a_7, i.e., $\{a_5, a_6\}$, has been pre-computed. Then the query $Q(a_7)$ can be answered by

accessing 2 cells associated with members in the cover $\{a_5, a_6\}$. Obviously, the query accessing cost is lowered.

In the partial pre-computation of 1-dimensional database, a cover is said to be *available* if every member in the cover is either a primary member or a pre-computed member. For any member m, there exists at least one available cover, which is the set of primary descendent members of m. If there is more than one cover available for a point query in a partially pre-computed 1-dimensional database, we should choose one with smallest cardinality, which is called *MinCover*, because the smaller the available cover is, the faster the query is answered as less cells will be accessed. A MinCover of a given point query in 1-dimensional database can be found by the following algorithm in case of partial pre-computation.

FindMinCover Algorithm:

Input: the dimension hierarchy of 1-dimensional database, a set of pre-computed members and any member m.

Output: C, a set of members

FindMinCover(m):

1) Initially, let C be the set of primary members that are also descendents of m, which is an available cover of m.
2) For each member, m_i, in C.
    a) Locate the path from m_i to m.
    b) Find the pre-computed member, m_i', which is closest to m in the path and add m_i' into C.
    c) Remove from C the members which are descendent of m_i'.

Lemma 3. If C and C' are two covers of a member m and every member in C' is a member in C or a descendent of some member in C, then $|C| \leq |C'|$.

Proof: Let C and C' be two sets of members, $\{m_1, m_2, \ldots, m_n\}$ and $\{m'_1, m'_2, \ldots, m'_l\}$. Divide C' into a set of n subsets, $\{s_1, s_2, \ldots, s_n\}$, such that each subset s_i contains m_i or all members in C' that are descendents of m_i in C. First, we will prove that $s_i \cap s_j = \emptyset$ if $1 \leq i \neq j \leq n$. Suppose there is a member m'_l in C' which is a descendent of two members, m_i and m_j, in C. Then, a primary descendent member of m'_l will be descendent of m_i and m_j too, which means that there is overlap between $pdm(m_i)$ and $pdm(m_j)$. This contradicts the definition of cover. Moreover, each subset is also non-empty, or else there will be some primary members that are descendents of a member in C but not descendents of any member in C', which also contradicts the definition of cover. Thus, we have $|C| \leq |C'|$. Q.E.D.

Theorem 1. Algorithm **FindMinCover** correctly outputs an available MinCover of any member m in the dimension hierarchy of 1-dimensional database in case of partial pre-computation.

Proof: Obviously, the output C is an available cover, since none of members in C can be a descendent of any other members in C. And C initially contains all the primary descendent members of m and a primary member will be removed from C only when its pre-computed ancestor has been added into C.

We are going to prove that the size of C is no larger than that of any other available cover of m in a partially pre-computed 1-dimensional database. Suppose there is another available cover of m, C'. According to the definition of cover, for any primary descendent member of m, p_i, either p_i is a member in C' or there is one member p' in C', which is the ancestor of p_i. Moreover, according to the FindMinCover Algorithm, either p_i is a member in C or there is one member p in C, which is the pre-computed ancestor of

p_i that is closest to m. Thus, p' is either p itself or a descendent of p. So, for available covers of m, C and C', any member in C' is either a member in C or a descendent of some member in C. According to Lemma 3, we have $|C| \leq |C'|$. Q.E.D.

So, the FindMinCover algorithm is able to find an available cover with smallest cardinality for any member in the dimension hierarchy of a 1-dimensional database in case of partial pre-computation. The time complexity of the FindMinCover algorithm is linear to the number of primary descendents a member has. We now formulate our query processing strategy for 1-dimensional databases, which is:

Query Processing in partially pre-computed 1-dimensional database:

1) [preparation] For any members in the dimension hierarchy, m_i, compute a MinCover of m_i.
2) [processing] For any point query $Q(m)$, retrieve the MinCover of m computed in step 1).
3) [accessing] Access the cells associated with members in MinCover of m and compute the answer of Q.

In the preparation phase of the above algorithm, we can compute a MinCover for every member in the dimension hierarchy, i.e. by applying the FindMinCover for each group member in the hierarchy and storing them in a table. A Mincover of any member can be retrieved during the processing phase. The above query processing strategy has a very low processing overhead because only one simple table lookup operation is required when a query is posed. This query processing strategy is also effective because the point query is computed from its MinCover, which is one of the smallest sets of members from which the answer of a given query can be computed.

3.2.4 Query Processing in Multidimensional Database

Intuitively, the query processing in 1-dimensional database can be extended for multidimensional database as the following:

For a point query $Q(m_1, m_2, ..., m_k)$,

1) find a MinCover for each member in the coordinate of Q: $MinCover(m_1)$, $MinCover(m_2)$, ..., $MinCover(m_k)$.

2) compute the answer of the query Q from the set of cells in the Cartesian product of MinCovers: $MinCover(m_1) \times MinCover(m_2) \times ... \times MinCover(m_k)$

However, this simple extension from 1-dimensional database is not always correct because there is no guarantee that the MinCovers for all members in the coordinate of Q exist and all the cells in the Cartesian product of MinCovers have been pre-computed. One method of making this query processing feasible is to make sure all cells in the above Cartesian product are pre-computed.

3.3 Cover-based Query Processing

In our scheme, we assume there is a subcube of the extended multidimensional space, every cell of which has been pre-computed. We call that subcube as Pre-computation Cube or PC Cube. Each side of PC Cube contains a subset of all members of that dimension, Selected Members Set, which is carefully chosen such that we can compute a MinCover for any member from the Selected Members Set of that dimension. So if we want to compute the answer of query $Q(m_1, m_2, ..., m_k)$, we will be able to use the Cartesian product of MinCovers: $MinCover(m_1) \times MinCover(m_2) \times ... \times MinCover(m_k)$ because all cells in that Cartesian product of MinCovers have been pre-

computed and included in the PC Cube. We will address the selection of members for the PC Cube in Chapter 4 and the computation of PC Cube in Chapter 5.

With the presence of the Pre-computation Cube, any given point query will be able to be processed by the following method:

Cover-based Query Processing

1) [preparation] Compute a MinCover of each member in every dimension,
2) [processing] For any point query $Q(m_1, m_2, ...,m_k)$, get a MinCover for each member in Q's coordinate.
3) [accessing] Access the set of cells in the Cartesian product of MinCovers:

$$MinCover(m_1) \times MinCover(m_2) \times \ldots \times MinCover(m_k)$$

In our strategy, the MinCover for each member in every dimension is computed and stored in tables in the preparation phase. Thus, in the processing phase, only k table lookup operations are required, where k is the number of dimensions. This gives us a very low processing overhead. In cover-based query processing method, the accessing cost of a query, which is defined as the number of cells accessed, is the cardinality of the above Cartesian product of MinCovers. We now will show that the accessing cost of a query is the lowest one, given the presence of PC Cube.

We define the *primary descendent cells* of a cell $T(m_1, m_2, ...,m_k)$, $pdc(T)$, as the set of all primary cells that are descendents of T. The primary descendent cells of T are, $pdm(m_1) \times pdm(m_2) \times \ldots \times pdm(m_k)$, the Cartesian product of the primary descendent members of members in T's coordinate. According to Lemma 1, the measure of a cell T can be computed from its primary descendent cells, $pdc(T)$.

Having defined the cover for a member in 1-dimensional database (section 3.2.3), we now extend the concept of cover into multidimensional database.

Definition 3.7. Multidimensional Cover of a cell T is a set of cells, $\{T_1, ..., T_n\}$, in extended multidimensional space if $\bigcup_{i=1}^{n} pdc(T_i)=pdc(T)$ and $pdc(T_i) \cap pdc(T_j) = \varnothing$ for $1 \le i \ne j \le n$.

Obviously, the measure of T, which is the result of a non-holistic aggregate function over $pdc(T)$, is computable from a multidimensional cover of T. In the partial pre-computation of multidimensional OLAP database, a multidimensional cover of a cell is said to be available if every cell in the multidimensional cover is either a primary cell or a pre-computed aggregation cell. For any cell in the extended multidimensional space, there always exists at least one available multidimensional cover, its primary descendent cells.

We will now prove that the set of cells used to answer a given point query in cover-based query processing is one available multidimensional cover with smallest cardinality, with the presence of PC Cube.

Lemma 4: Suppose S and S' are two multidimensional covers of a cell T. If every cell in S' is a cell in S or a descendent of some cell in S, then $|S| <= |S'|$

Proof: Let S and S' be two sets of cells, $\{c_1, c_2, ..., c_n\}$ and $\{c'_1, c'_2, . ., c'_l\}$. First, we will prove that if a cell in S' is a descendent of some cell in S, then it cannot be a descendent of more than one cell in S. Suppose that a cell c' in S' is a descendent of two cells, c_1 and c_2, in S. Then primary descendent cells of c' are descendents of c_1 and c_2, which means that there is overlap between $pdc(c_1)$ and $pdc(c_2)$. This contradicts the definition of the multidimensional cover.

Now, let's divide S' into a set of n subsets, $\{s_1, s_2, ..., s_n\}$, such that each subset s_i contains c_i or all cells in S' that are descendents of s_i. Each subset will have no overlap with any other subsets because a cell in S' can only be a descendent of one cell in S. Each subset is also non-empty, or else there will be some primary cells that are descendents of a cell in S but not descendents of any cell in S', which contradicts the definition of multidimensional cover. Thus, we have $|S| \leq |S'|$. Q.E.D.

Theorem 2. With the presence of PC Cube, for any point query $Q(m_1, m_2, ..., m_k)$, the set of cells in the Cartesian product of the MinCovers of the members in the Q's coordinate is one of the smallest available multidimensional covers of Q.

Proof: We first prove that for any point query $Q(m_1, m_2, ..., m_k)$, the set of cells in the Cartesian product of the MinCovers of the members in Q's coordinate, is an available multidimensional cover of Q. Let S denote the Cartesian product of the MinCovers of the members in Q's coordinate. Suppose $T_i(t_{i1}, t_{i2}, ..., t_{ik})$ is a cell in S, where t_{ij} is the j-th member in the coordinate of T_i and $t_{ij} \in MinCover(m_i)$ for $1 \leq i \leq k$. Then, the primary descendent cells of T_i are $pdm(t_{i1}) \times pdm(t_{i2}) \times ... \times pdm(t_{ik})$. According to the definition of MinCover, it is easy to prove that the union of $pdc(T_i)$ is equal to the $pdc(Q)$, and there is no intersection between the sets of primary descendent cells of any two cells in S. Thus, for point query Q, the Cartesian product of the MinCovers of the members in the coordinate of Q, is a multidimensional cover of Q. Obviously, every cell in S is included in PC Cube, which means S is available.

We now prove that S is one of the smallest available multidimensional covers of Q. Assume there exists another available multidimensional cover of Q in PC Cube, S'. Obviously, the union of primary descendent cells of cells in S' should be equal to the set of primary descendent cells of Q. Suppose a primary cell $P(p_1, p_2, ..., p_k)$ is a descendent of $Q(m_1, m_2, ..., m_k)$. There must exist a cell $T'(t'_1, t'_2, ..., t'_k)$ in S', such that

P is either T' or a primary descendent of T'. The i-th member in the coordinate of T', t'_i, is then p_i or an ancestor of p_i. Meanwhile, T' is a descendent of Q, so t'_i is either m_i or descendent of m_i. Thus, t'_i is a member in the path from p_i to m_i. According to the definition of FindMinCover, there exists a member in the *MinCover*(m_i), which is the member closest to m_i in the path from p_i to m_i. So, for any member t'_i in the coordinate of T', there exists a member t_i in *MinCover*(m_i), which is either t'_i or an ancestor of t'_i. Then there always is a cell in the Cartesian product of *MinCover*(m_i), which is T' or an ancestor of T'. And we already have S is the such Cartesian product of *MinCover*(m_i). As a result, for any cell T' in S', there is a cell T in S, which is either T' or an ancestor of T'. According to Lemma 4, we have $|S| \leq |S'|$. Q.E.D.

To conclude this chapter, we devise a cover-based query processing strategy at the presence of PC Cube. Our strategy has a low processing overhead and can use a set of cells with smallest cardinality to answer any given point query, which gives us the lowest accessing cost.

CHAPTER 4

MEMBER SELECTION ALGORITHM

In the section 3.3, we introduced a new organization of pre-computed cells, PC Cube, which is crucial to our query processing strategy. Each dimension of the PC Cube contains a subset of members in that dimension. In this chapter, we will present our methods of selecting members for inclusion into PC Cube.

4.1 Member Selection Algorithm -- Greedy

The motivation for developing Member Selection Algorithm (MSA) -- Greedy is to find a member-based version of greedy algorithm in [HRU96]. The greedy algorithm in [HRU96] chooses the "best" view for materialization, one at a time.

Following a similar approach, our Member Selection Algorithm -- Greedy is to look for a member for selection that would achieve maximum benefit per unit of storage, on the basis of the additional amount of storage that would be taken up by the pre-computation. A sketch of MSA-Greedy would look like:

Member Selection Algorithm ---Greedy (Sketch)

Input: Dimension hierarchy for each dimension and space limit

Output: PC Cube formed by the selected members set on each dimension

k: Number of dimensions

1) Initially, let the PC Cube be the primary database, and selected members set in each dimension be all the primary member of that dimension
2) Repeat
    a) Find the member with highest benefit per unit storage among all dimensions, say m_j.
    b) Include m_j as a new member of the *j*-th dimension in the PC Cube

Until PC Cube is larger than a pre-set space limit.

Obviously, we have to compute the amount of storage the pre-computation would take up, and the benefit that the selection of a certain member would bring.

Suppose we use $Cube(SMS_1, \ldots, SMS_i, \ldots, SMS_k)$ to denote the PC Cube, where SMS_i is the set of selected members in the *i*-th dimension. If a member in the *i*-th dimension, say *M*, is added as a new member in the *i*-th dimension of the PC Cube, then the new PC Cube becomes $Cube'(SMS_1, \ldots, SMS'_i, \ldots, SMS_k)$, where $SMS_i' = SMS_i \cup \{M\}$. Thus, the amount of the additional storage required is the difference between the sizes of two cubes, *Cube* and *Cube'*. The size of a cube may be estimated by using a method in [SDNR96].

The benefit of the pre-computation is equal to the reduction in the accessing cost of all possible queries in the query workload. Our query workload, as defined in section 1.7.2, is a set of all point queries in the extended multidimensional space. The accessing cost of a query is defined to be the number of cells that must be accessed in order to compute the answer of the query. Suppose *Q* is a query in the workload. Let *C*(*Q*) be the accessing cost of answering the query *Q*. The number of cells that must be accessed to

answer the query Q is the size of the Cartesian product of MinCovers of members in Q's coordinate according to our query processing method. When an additional member is selected for inclusion into the PC Cube, the MinCovers of some members in Q's coordinate may be different. Let $C'(Q)$ be cardinality of Cartesian product of new MinCovers. Thus, the reduction of accessing cost for answering Q is $C(Q)$-$C'(Q)$. The total reduction of the accessing cost, also the benefit, is $\sum C(Q)-C'(Q)$ over all possible queries in the query workload. The benefit per unit storage is then the benefit divided by the additional storage space required. In order to gain an appreciation of how the benefit may be calculated, we will start with an example in the following section.

4.2 Example Benefit Analysis

In this benefit analysis, we use a very simple 2-dimensional database in Section 3.2.2, which has two dimension attributes, dimension A and B. Both dimension A and B consists of 7 members. Their dimension hierarchies are shown in Figure 3.2. In the following table (Table 4.1), we show the accessing cost of all point queries in case of no pre-computation. For example, the accessing cost of query (a_5, b_5) is 4, since 4 cells in $\{a_1, a_2\} \times \{b_1, b_2\}$ need to be accessed.

	b_1	b_2	b_3	b_4	b_5	b_6	b_7
a_1	1	1	1	1	2	2	4
a_2	1	1	1	1	2	2	4
a_3	1	1	1	1	2	2	4
a_4	1	1	1	1	2	2	4
a_5	2	2	2	2	4	4	8
a_6	2	2	2	2	4	4	8
a_7	4	4	4	4	8	8	16

Table 4.1 Cost of answering all queries -- no pre-computation

Let's consider the accessing cost when b_5 has been selected for pre-computation. Table 4.1 is then updated into Table 4.2, which shows the new accessing costs of all point queries.

	b_1	b_2	b_3	b_4	b_5	b_6	b_7
a_1	1	1	1	1	***1***	2	***3***
a_2	1	1	1	1	***1***	2	***3***
a_3	1	1	1	1	***1***	2	***3***
a_4	1	1	1	1	***1***	2	***3***
a_5	2	2	2	2	***2***	4	***6***
a_6	2	2	2	2	***2***	4	***6***
a_7	4	4	4	4	***4***	8	***12***

Table 4.2 Cost of answering all queries -- After the pre-computation of b_5

Note that only the columns b_5 and b_7 need be updated, i.e., all queries (*, b_5) and (*, b_7), where * stands for all members in the dimension A. For example, only 2 cells need to be accessed to answer the query (a_5, b_5) now, i.e., $\{a_1, a_2\} \times \{b_5\}$. For query ($a_5$, b_7), only 6 cells need to be accessed, i.e. $\{a_1, a_2\} \times \{b_3, b_4, b_5\}$. The benefit of pre-computation of b_5 is the sum of the reduction in costs of answering (*, b_5) and (*, b_7) queries.

Suppose a_5 is the next choice for selection. This time, the rows a_5 and a_7 in Table 4.2 are to be updated. The benefit of pre-computation of a_5 can be calculated accordingly (Table 4.3).

	b_1	b_2	b_3	b_4	b_5	b_6	b_7
a_1	1	1	1	1	***1***	2	***3***
a_2	1	1	1	1	***1***	2	***3***
a_3	1	1	1	1	***1***	2	***3***
a_4	1	1	1	1	***1***	2	***3***
a_5	***1***	***1***	***1***	***1***	***1***	***2***	***3***
a_6	2	2	2	2	***2***	4	***6***
a_7	***3***	***3***	***3***	***3***	***3***	***6***	***9***

Table 4.3 Cost of answering all queries -- After the pre-computation of b_5 and a_5

According to our cover-based query processing method, a point query is computed from the Cartesian product of the MinCovers of the members in the query's coordinate. So, at any point of this selection process, the total accessing cost of answering all point queries is:

$\sum_i \sum_j |MinCover(a_i) \times MinCover(b_j)| = (\sum_i |MinCover(a_i)|) * (\sum_j |MinCover(b_j)|)$, where i and j range from 1 to 7 respectively.

The expression is valid because the MinCover of a member in a particular dimension hierarchy is independent of the contents of any other dimensions. Each time a new member is selected and included into the PC Cube, the MinCovers of some members in that dimension are updated. We can compute the reduction in the total accessing cost of answering all queries, or equivalently, the benefit of pre-computation of a member in dimension A, which is

$(\sum_i |MinCover(a_i)| - \sum_i |MinCover'(a_i)|) * (\sum_j |MinCover(b_j)|)$, where $MinCover'(a_i)$ is the new MinCover for the members in dimension A after a member is selected.

The above example shows how to calculate the benefit of the selection of a member in 2-dimensional database. In next section, we will consider how to calculate the benefit of the members in multidimensional database, so that we can greedily select the 'best' member.

4.3 Benefit Calculation

We now present a general method to calculate the benefit of inclusion a new member for pre-computation. Suppose in certain stage of the selection process, we have

a PC Cube, which is the Cartesian product of the selected members set in each dimension:

$P_1 = SMS_1 \times \ldots \times SMS_i \times \ldots \times SMS_k$, where SMS_i is the selected members set in the i-th dimension (the initial value of SMS_i is the set of all primary members in the i-th dimension).

The cost of answering all possible point queries is:

$C_1 = (\Sigma|MinCover(m_1)|) * \ldots * (\Sigma|MinCover(m_i)|) * \ldots * (\Sigma|MinCover(m_k)|)$, where m_i ranges over all members of the i-th dimension hierarchy

Without losing generality, suppose a member in the 1^{st} dimension, say M, is added as a new member of the PC Cube, then the PC Cube becomes

$P_2 = (SMS_1 \cup \{M\}) \times \ldots \times SMS_i \times \ldots \times SMS_k$

The new cost of answering all possible point queries is:

$C_2 = (\Sigma|MinCover'(m_1)|) * \ldots * (\Sigma|MinCover(m_i)|) * \ldots * (\Sigma|MinCover(m_k)|)$, where m_i ranges over all members of the i-th dimension hierarchy and $MinCover'(m_1)$ is the new MinCover for the members in the 1^{st} dimension after M is selected.

We define the benefit of member as the saving in the total accessing cost of answering all point queries, and the benefit per unit space as the benefit of a member divided by the number of cells added:

$$B(M) = \frac{C_1 - C_2}{|P_2| - |P_1|}$$

$$= \frac{\Sigma|MinCover(m_1) \times \ldots \times MinCover(m_k)| - \Sigma|MinCover'(m_1) \times \ldots \times MinCover(m_k)|}{|\{M\} \times SMS_2 \times \ldots \times SMS_i \times \ldots \times SMS_k|}$$

$$= \frac{(\sum|MinCover(m_1)|-\sum|MinCover'(m_1)|)*(\sum|MinCover(m_2)|)*...*(\sum|MinCover(m_k)|)}{|SMS_2|*...*|SMS_i|*...*|SMS_k|}$$

$$= (\sum|MinCover(m_1)|-\sum|MinCover'(m_1)|)*\frac{\sum|MinCover(m_2)*...*\sum|MinCover(m_k)|}{|SMS_2|*...*|SMS_i|*...*|SMS_k|}$$

$$= (\sum|MinCover(m_1)|-\sum|MinCover'(m_1)|)*\prod_{i=2}^{k}\frac{\sum|MinCover(m_{ij})|}{|SMS_i|}$$

From the above formula, we can find that the member, which gives the maximum ($\sum|MinCover(m_1)|$ - $\sum|MinCover'(m_1)|$), has the greatest benefit among the unselected members in the 1^{st} dimension. Let's consider how to calculate ($\sum|MinCover(m_1)|$ - $\sum|MinCover'(m_1)|$). When a new member M in the 1^{st} dimension is selected, only the MinCovers of M and the consecutive un-selected parents of M in the 1^{st} dimension will be affected. We have the

$$MinCover'(m_1) = \begin{cases} 1 & \text{if } m_1 \text{ is } M; \\ MinCover(m_1) - MinCover(M) + 1, & \text{if } m_1 \text{ is one of the consecutive un-selected parents of } M. \\ MinCover(m_1) & \text{others} \end{cases}$$

So, $(\sum|MinCover(m_1)| - \sum|MinCover'(m_1)|) = (n+1)*(|MinCover(M)| - 1)$, where n is the number of consecutive un-selected parents of M.

Therefore, $B(M) = \frac{C_1-C_2}{|P_2|-|P_1|}$

$$= (n+1)\times(|MinCover(M)|-1)*\prod_{i=2}^{k}\frac{\sum|MinCover(m_{ij})|}{|SMS_i|},$$

where n is the number of consecutive un-selected parents of M, SMS_i is the selected members set in the i-th dimension, and m_{ij} ranges over all members in the i-th dimension.

We now state our member selection algorithm.

Member Selection Algorithm ---Greedy

Input: Dimension hierarchy for each dimension and space limit

Output: PC Cube formed by the selected members set on each dimension

k: Number of dimensions

3) Let the initial selected members set in each dimension be all the primary member of that dimension

4) For $i = 1$ to k, Compute a MinCover of each member in i-th dimension

5) Repeat

   c) For $i = 1$ to k

      Find the m_i, among the un-pre-computed members in the i-th dimension hierarchy, which will give the highest benefit per unit storage

   d) Find the member with highest benefit per unit storage among all dimensions, say m_j.

   e) Include m_j as a new member of the j-th dimension in the PC Cube

   f) Update cover of m_j and all relevant parents of m_j

Until PC Cube is larger than a pre-set space limit.

The step 2) of above algorithm is to compute the MinCover of each member by using the FindMinCover algorithm in section 3.2.3, whose time complexity is linear to the number of members in each dimension and which can be performed in advance. The step 3) is to greedily choose members for inclusion into the PC Cube and is the major loop of above algorithm. The time complexity of our Greedy Member Selection algorithm is $O(k*n*l)$, where k is the number of dimensions, n is the average number members in each dimension and l is the number of members selected.

4.4 Variations of Member Selection Algorithm

Besides greedy selection of members, we have other approaches to choose the members on each dimension. We introduce some variations of the member selection algorithm in this section.

4.5.1 MSA --- Bottom Up

In this approach, all primary members in each dimension are included into the selected members set of that dimension. This selection simply chooses the group members from the lowest level in the dimension hierarchy. The Figure 4.1 and 4.2 show an example dimension hierarchy with selected members, and the PC Cube.

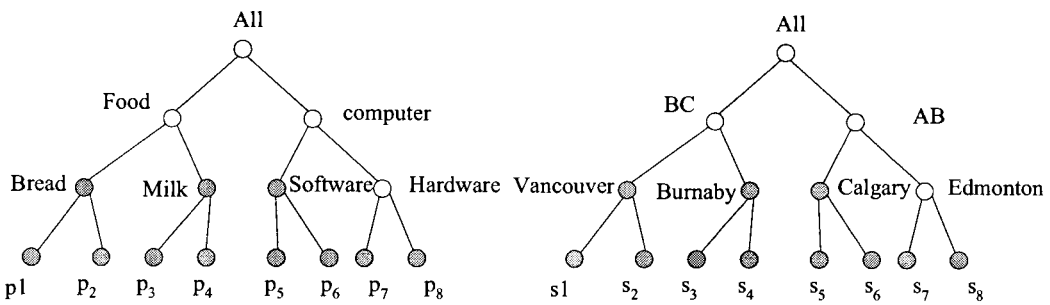


Figure 4.1 Nodes in shadow are selected members in dimension *Product* and *Location*

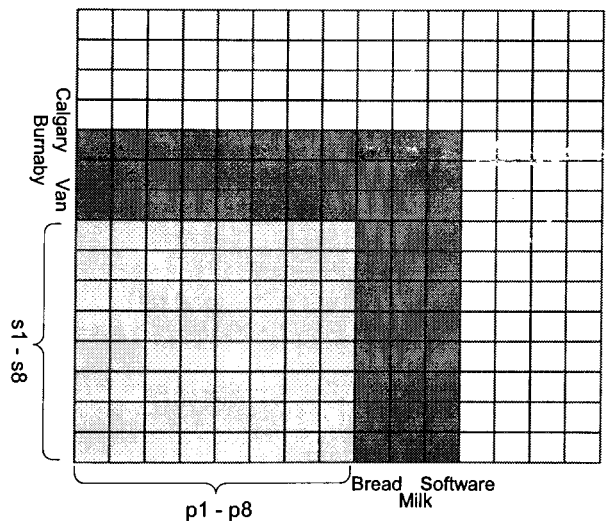


Figure 4.2 The area in shadow is the resulting PC Cube by MSA-BottomUp

4.5.2 MSA --- Top Down

Top down approach is an opposite of the bottom up approach. The members are chosen starting from the highest level in the dimension hierarchy. The example dimension hierarchy, the selected members and PC Cube can be depicted as the following figure (Figure 4.3).

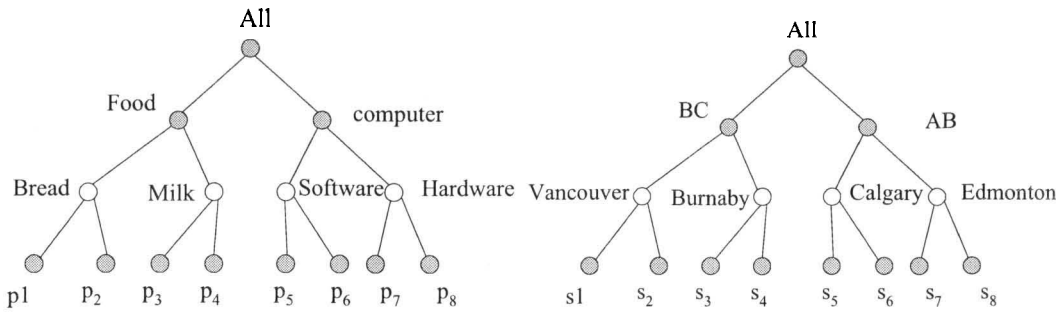


Figure 4.3 Nodes in shadow are selected members in dimension *Product* and *Location*

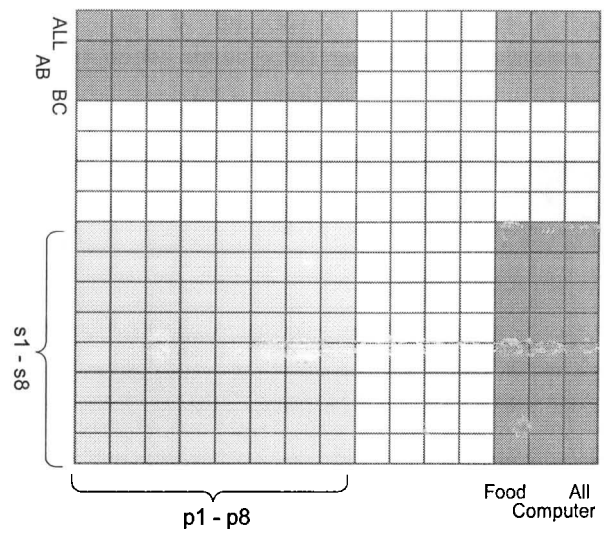


Figure 4.4 The area in shadow is the resulting PC Cube by MSA-TopDown

4.5.3 MSA --- No Redundancy

In the member selection methods mentioned before, the primary members in a dimension are always included as a part of the selected members in that dimension. There is another variation. When a group member is selected, its primary children members will be removed from the selected members set of a perspective dimension. We call this selection method as MSA – No Redundancy. An example of the dimension hierarchies with the selected members and the PC Cube is shown as the following [Figure 4.5 and Figure 4.6].

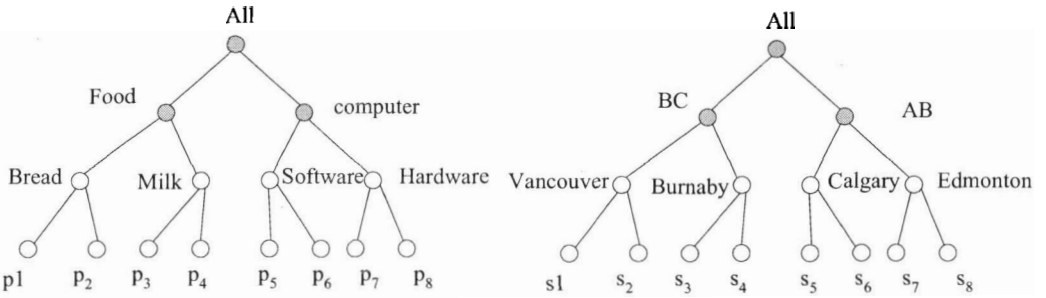


Figure 4.5 Green nodes are selected members in dimension *Product* and *Location*

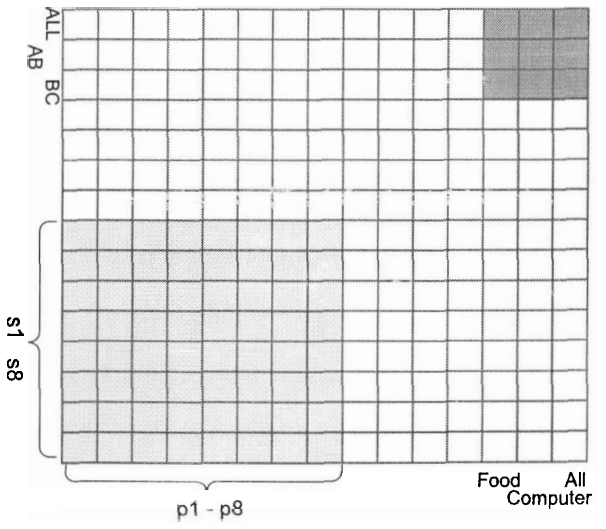


Figure 4.6 The area in shadow is the resulting PC Cube by MSA-NoRedundancy

Obviously, some other approaches to build the PC Cube are possible. For example, the selected members set for the PC Cube can be any subset of the set all members in a dimension. The selected members do not have to always contain all primary members in that dimension. Currently, we don't consider more variations of member selection algorithm.

CHAPTER 5

IMPLEMENTATION AND EXPERIMENTATION

In this chapter, we first examine the computation of the PC Cube. Then we will briefly introduce the implementation of our approaches. The experimentation is shown afterwards.

5.1 Computation of PC Cube

Basically, we use ADODA ([Luk01]) to compute the PC Cube. The ADODA summation algorithm is proposed mainly for the full pre-computation and to compute all aggregation cells in the extended multidimensional space. It cannot be directly deployed to compute the PC Cube, whose dimensions contain only the selected members. However, we find that the desired PC Cube can be obtained by a full pre-computation over the "modified" dimension hierarchies.

Let's use the example of 2-dimensional database in section 4.5.2. Their dimension hierarchies with the selected members are shown in Figure 5.1. We can

modify the dimension hierarchies of *Product* and *Location* and leave only the selected members (Figure 5.2).

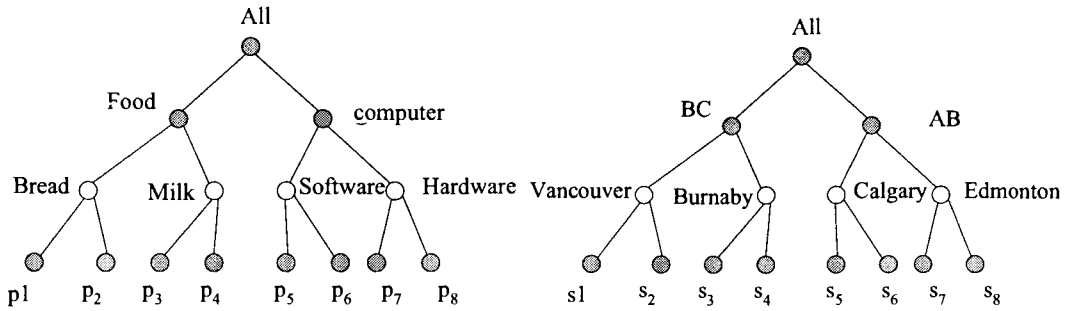


Figure 5.1 Dimension hierarchies of *Product* and *Location*

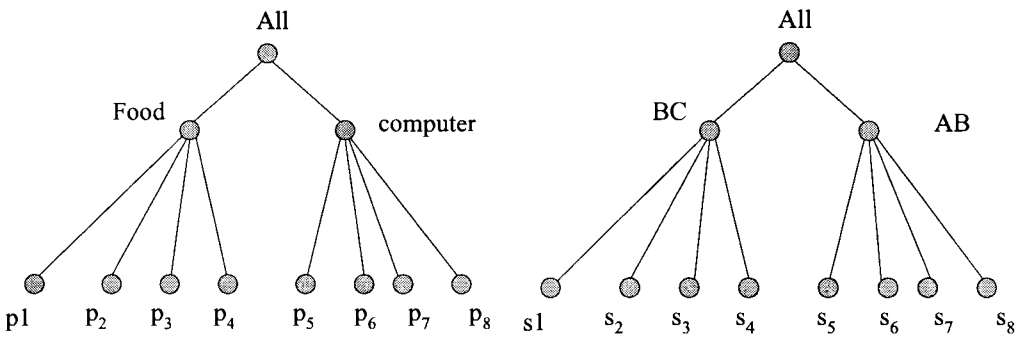
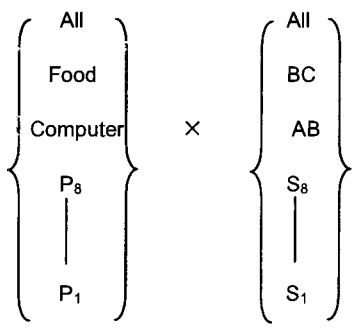


Figure 5.2 Modified dimension hierarchies with only the selected members

A full pre-computation over the "modified" dimension hierarchies can be noted as the following Cartesian product.



The above Cartesian product exactly defines a PC Cube whose dimension contains only the selected members. So, we are able to apply the ADODA on modified dimension hierarchies to compute the PC Cube.

5.2 The Architecture of Our Implementation

We have introduced our approaches in selection, pre-computation and query processing. In this section, we will present the architecture of our implementation. Our architecture has five components: Selection Manager, Computation Manger, Cube Manager, Storage Manger, and Query Manager, which are interacted with each other as shown in Figure 5.3.

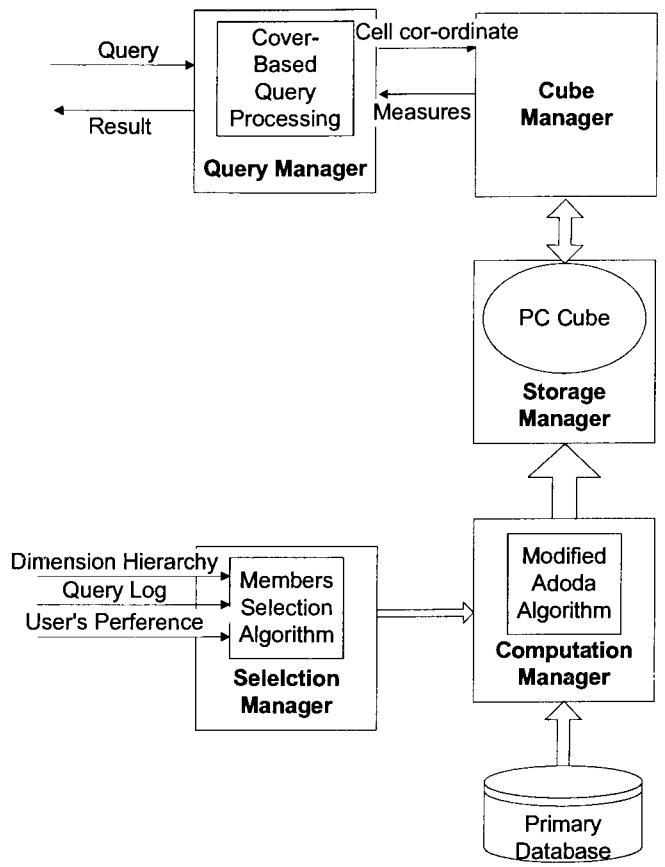


Figure 5.3 Our implementation framework

1. Selection Manger: Its main task is to generate the selected members set for each dimension of the PC Cube, which will be used for the pre-computation. The

selection manager can accept the dimension hierarchy as the input as well as the query log and users' preference. Thus, the selection manager can generate the selected member set, which is fine tuned according to the query log and users' preference.

2. Computation Manager: It uses the modified ADODA Summation algorithm to generate the Pre-computation Cube defined by the selected members set obtained from the selection manager.

3. Storage Manger: It takes care of the storage of the PC Cube generated by the Computation Manager. It also handles the organization and the compression of the PC Cube.

4. Cube Manager: It mainly takes care of the retrieval of the pre-computed cells from the PC Cube. It accepts the retrieval requests issued from the Query Manager and returns the measures of the cells.

5. Query Manager: It is the main module to handle the users' queries. Each user's query is processed by the cover-based query processing method, decomposed into a set of pre-computed cells in the PC Cube. After the Query Manager sends the retrieval requests to and get the desired measures back from the Cube Manager, the result is then formed and returned to the user.

5.3 Experimentation

The objective of the experimentation is to examine the feasibility of cover-based query processing method and the partial pre-computation scheme. We also compare the effectiveness of the various member selection algorithms.

5.3.1 Experiment Setup

Test data set

The test data set serves as the primary database, which contains all the primary tuples and is used to compute the aggregation tuples. In our experimentation, we use two data sets. The first one is a real data set that represents 5 years of enrolment data in SFU. The schema of this data set has 5 dimension attributes and 2 measure attributes. And there are about 20000 primary tuples in the fact table and about 5 million aggregation tuples. The second data set we use is a synthetic data set, whose scheme is adopted from the one in [SDN98]. The second data set has 5 dimensions, but its dimension hierarchies have more members on average and are more complicated than that of SFU enrolment data set. Each dimension of the second data set's schema has 6 levels and has on each level {100, 50, 25, 5, 2, 1} members, starting from the bottom level. The second data set allow us to test the scalability of our approach with respect to the number of aggregation cells generated and the complexity of the dimension hierarchy. This test data set has 100,000 primary cells, which are uniformly distributed. The number of all possible aggregation cells that can be generated from the second data set is about $3.6*10^8$.

Test queries set

Queries are randomly generated from the query workload we study in this thesis, i.e., all point queries in the extended multidimensional space. Two experimental parameters are available: the number of queries and the number of group members in the coordinate of point query. The number of group members in the coordinate of query shows the aggregation level of the query. The more group members in a query, the more aggregation is required. Through the test over different test queries, we can tell how the different selection schemes influence the query performance for queries at different

aggregation levels. In our experiments, we have 6 test query sets. The first 5 test sets are 10,000 random point queries with 1-5 group members in their coordinates respectively. The 6^{th} test query set has 50,000 point queries, which is the union of the first 5 test query sets.

Partial pre-computation

Currently, we implement 5 partial pre-computation schemes: MSA – Greedy, MSA --- Top down, MSA --- Bottom up, MSA --- No redundancy, and MSA – Random. MSA-Random is to randomly choose the group members and is included for the comparison of query performance with MSA-Greedy.

In our approach, we can specify the pre-computation ratio, which ranges from 0% to 100%. We are able to examine the query performance of selection schemes at different pre-computation ratio.

Time measurement

Processing overhead: the time spent in determining the set of cells that should be accessed in answering a given point query.

Accessing cost: the time spent in accessing the set of cells determined by a query processing, which is related to the number of cells that must be accessed in order to compute the answer of a given point query.

Query answering time: the sum of processing overhead and the accessing cost of a point query.

Test machine

Out experiments are performed on an Intel Pentium IV machine running Windows 2000. The machine has a physical memory of 1.5G bytes.

System status

We assume a warm system in our tests, which is that all pre-computed cells have been loaded into memory.

5.3.2 Experimental results

Experiment A: Processing Overhead

We examine the processing overhead, which is the time spent in determining the set of cells required to be accessed. Table 5.1 and Table 5.2 show the processing overhead of 10,000 point queries with different aggregation levels.

Query Complexity (# of group members in query)	Pre-computation Ratio 10%	Pre-computation Ratio 20%
1	17ms	21ms
3	22ms	20ms
5	19ms	22ms

Table 5.1 The processing overhead for 10,000 point queries (SFU enrolment data set)

Query Complexity (# of group members in query)	Pre-computation Ratio 10%	Pre-computation Ratio 15%
1	16ms	22ms
3	21ms	25ms
5	23ms	25ms

Table 5.2 processing overhead for 10,000 point queries (synthetic data set)

From Table 5.1 and Table 5.2, we find that the query processing overhead on two data sets are almost the same though the sizes of two data sets and the numbers of aggregation cells generated are quite different. There is also no obvious change for queries with different aggregation. The reason is that the processing overhead in our query processing method is only related to the number of dimensions of a relation.

We also examine the percentage of processing overhead over query answering time for 10,000 point queries on different aggregation levels, which has been shown in the following tables (Table 5.3 and 5.4).

Query Complexity (# of group members in query)	Pre-computation Ratio 10%	Pre-computation Ratio 20%
1	33%	29%
3	7.9%	16%
5	2.3%	11%

Table 5.3 The percentage of query processing overhead over query answering time (SFU enrolment data set)

Query Complexity (# of group members in query)	Pre-computation Ratio 10%	Pre-computation Ratio 15%
1	27.1%	34.3%
3	23.6%	29.4%
5	12.3%	20.2%

Table 5.4 The percentage of query processing overhead over query answering time (synthetic data set)

We observe that the relative processing overhead over the cost of answering queries is significant, but not high. We notice that the relative cost decrease quickly when the set of test queries requires more aggregation. It is also interesting to note that the relative cost increases as the pre-computation ratio increases. This is due to the fact that the processing time has little change, but the total answering time is reduced when more pre-computation has been done.

Experiment B: Accessing Cost

In this experiment, we mainly examine the accessing cost of queries at different pre-computation ratio. The accessing cost is calculated on the total number of cells accessed for answering all point queries in a test query set. The following graphs (Figure 5.4 and Figure 5.5) show the number of cells accessed for 10,000 point queries with 1

and 2 group members at different pre-computation ratio on two data sets. The selection scheme is MSA---Greedy.

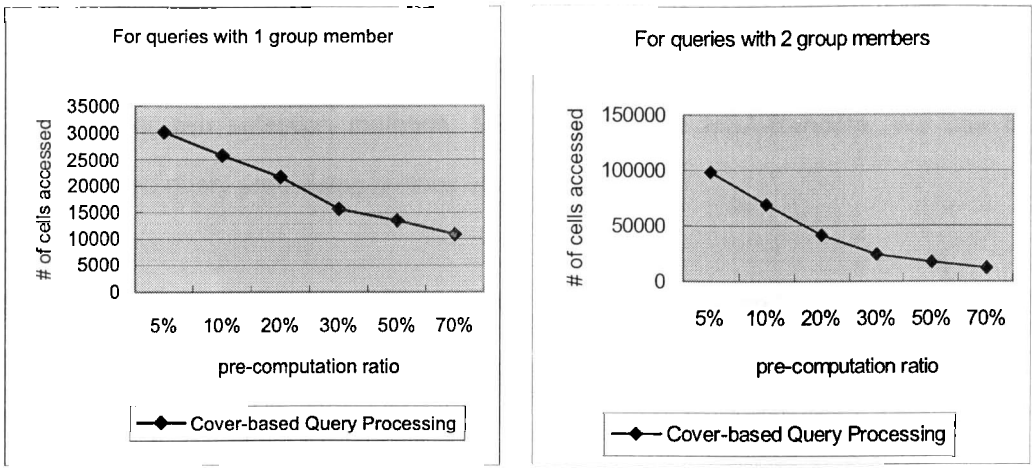


Figure 5.6 Number of cells accessed for 10,000 point queries (SFU enrolment data set)

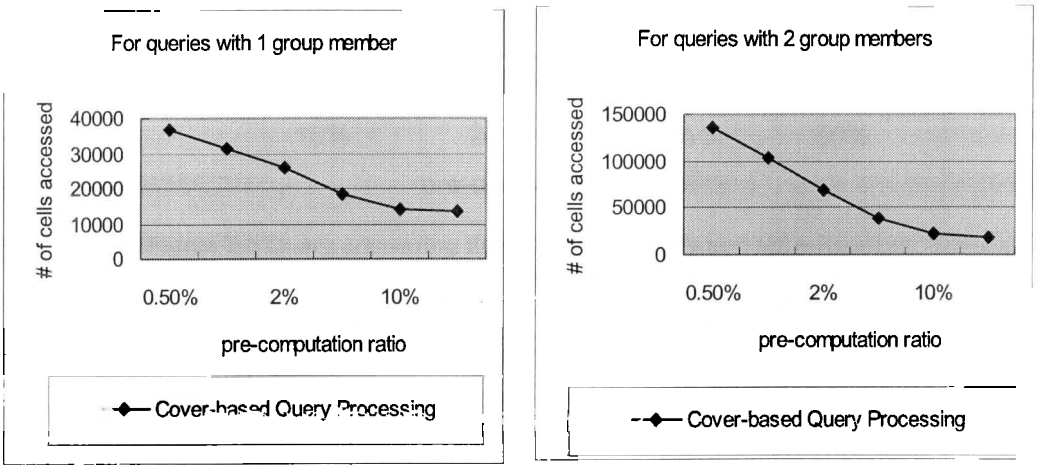


Figure 5.7 Number of cells accessed for 10,000 point queries (synthetic data set)

From the above graphs, we observe that much fewer cells are accessed to answer point queries with the increase of partial pre-computation ratio. This experiment shows that our member selection algorithm is effective, and able to choose the cells that would reduce the total accessing cost. It also reflects that our cover-based query processing method is able to make use of the pre-computed cells.

Experiment C: MSA -- Greedy Vs. MSA -- Random

Another way of showing the effectiveness of the MSA-Greedy is to compare it with an algorithm that randomly selects members for inclusion into PC Cube. In this experiment, we compare the query answering time on test query set 6 (50,000 point queries) for two selection methods: MSA–Greedy and MSA-Random. We use the cover-based query processing method in this experiment.

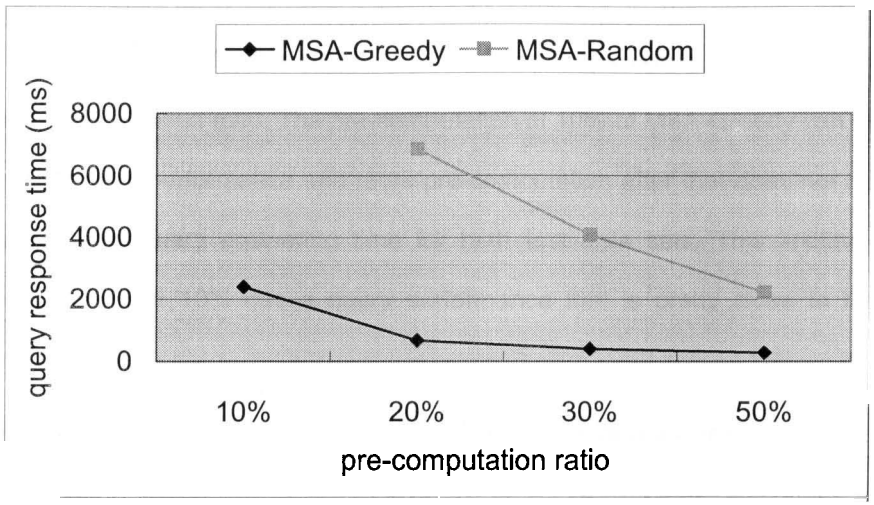


Figure 5.8 Query answering time for MSA-Greedy and MSA-Random (SFU enrolment data set)

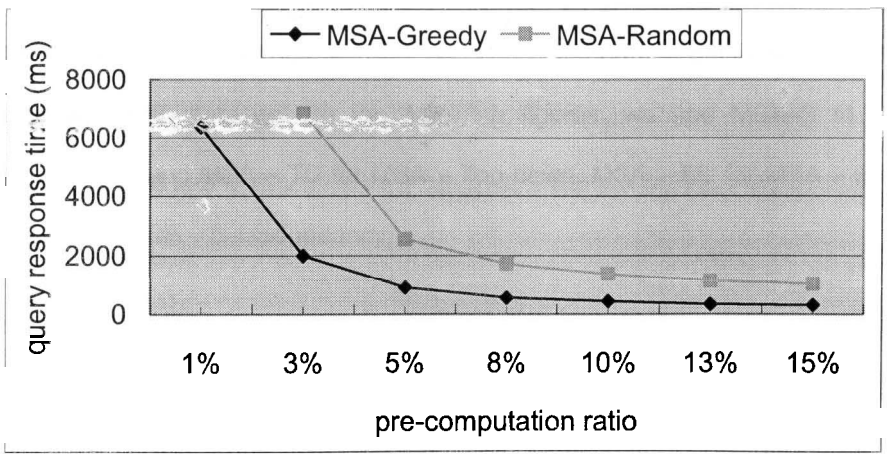


Figure 5.9 Query answering time for MSA-Greedy and MSA-Random (synthetic data set)

From the above graphs (Figure 5.6 and Figure 5.7), we observe that both the MSA-Greedy and MSA-Random can yield a better query performance when more pre-computation has been done. But, the MSA-Greedy outperforms the MSA-Random at all pre-computation ratio. Even the difference between the query answering time for two selection methods shrinks with the increase of pre-computation ratio, the query performance for MSA-Greedy is still at least 3 times better than that for MSA-Random at a high pre-computation ratio (50% for SFU and 15% for synthetic data set).

For MSA -Greedy, the query answering time drops very quickly with the increase of the pre-computation ratio. The pre-computation at 10% of MSA-Greedy can produce a fairly good query performance and more pre-computation after that does not help much in reducing the query answering time for both test data sets. This implies that the pre-computation at 10% has a query performance that is pretty close to that of full pre-computation.

Experiment D: Comparison of various member selection algorithms.

In this experiment, we study the impact of different selection methods to the query performance for queries at different aggregation level. We collect the query answering time for 10,000 queries with 1-5 group members at different pre-computation ratio on synthetic data set. In the following figures, we use MSA-G to represent MSA – Greedy, and MSA – TD for MSA – Top down, MSA – BU for MSA – Bottom up, MSA – NR for MSA – No redundancy.

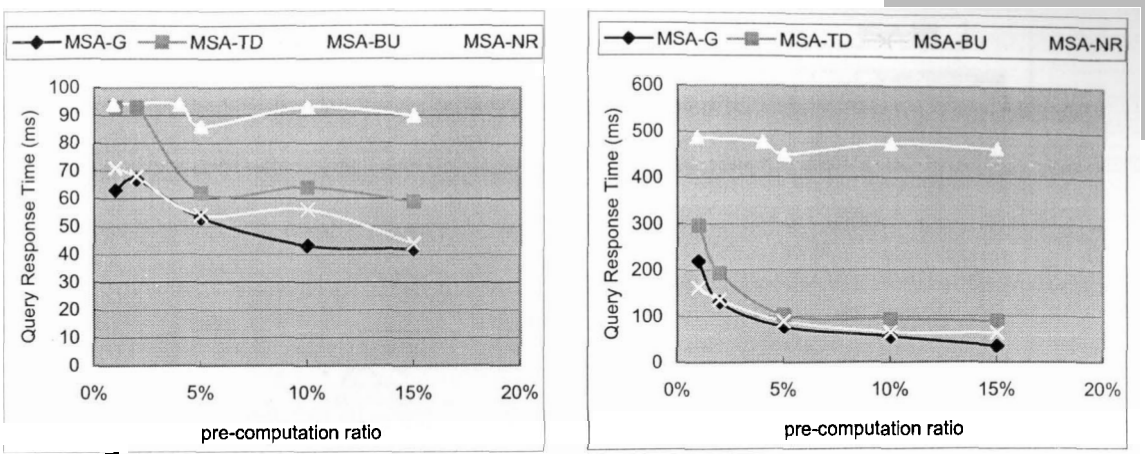


Figure 5.10 Query answering time for queries with 1 and 2 group Members (synthetic data set)

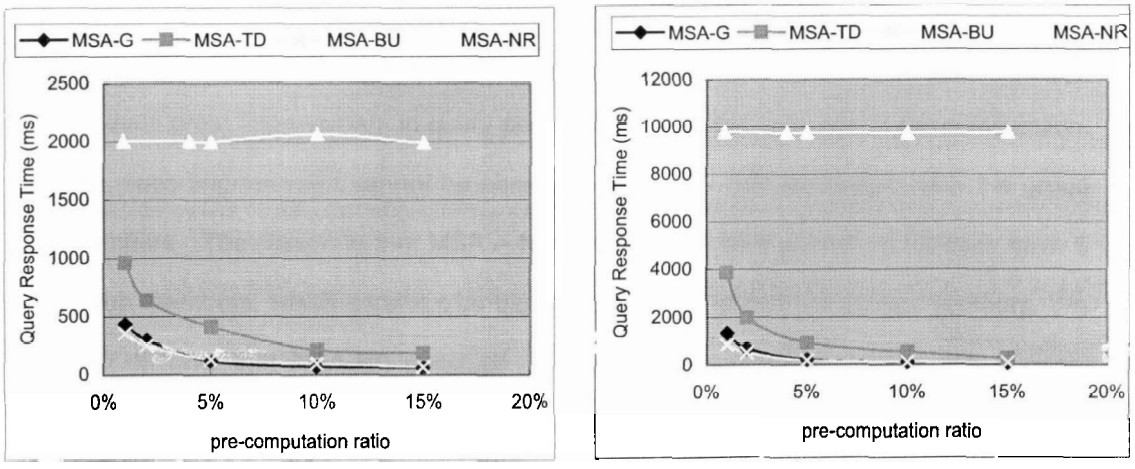


Figure 5.11 Query answering time for queries with 3 and 4 group Members (synthetic data set)

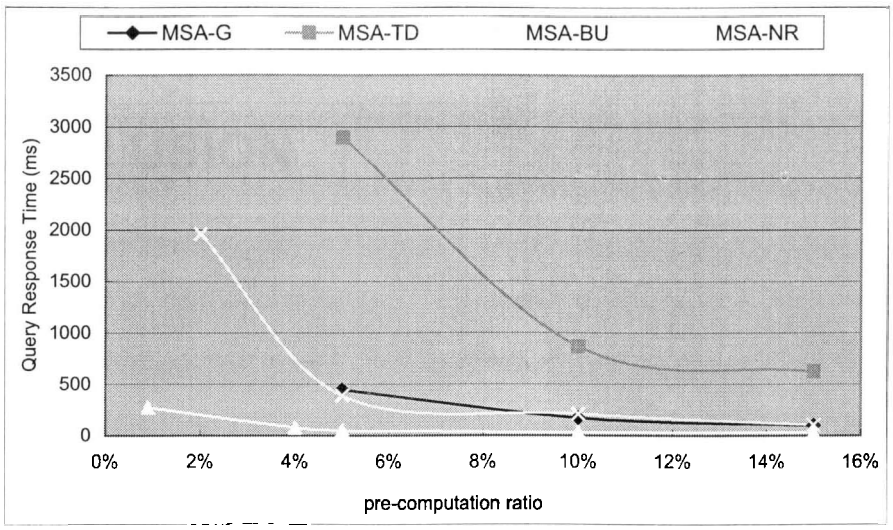


Figure 5.12 Query answering time for queries with 5 group Members (synthetic data set)

For queries at all aggregation levels, MSA-G, MSA –TD, and MSA –BU can produce great improvement in query performance with the increase of pre-computation. But, such improvement cannot be observed for MSA – NR for queries with 1-4 group members. The reason is that MSA – NR chooses the cells whose coordinates have 5 group members, which cannot give benefit to queries with 1 – 4 group members. We also observe that MSA-NR gives us the best performance for queries with 5 group members. Other selection algorithms, however, have a much better overall performance.

CHAPTER 6

CONCLUSION

6.1 Summary

The partial pre-computation is a popular topic in the OLAP research. However, most of the published papers for partial pre-computation are about optimization for views as the query workload. In this thesis, we present a new approach to this research topic, by studying the optimisation problem based on a different workload, i.e., the point queries in extended multidimensional space, which includes range queries over the primary database. We argue that this query workload is representative of queries with small answers, and is well suited for the MOLAP systems that features point style accessing at the lowest storage level.

Most works on partial pre-computation do not consider the processing overhead, which is the time spent in determining how to answer a given query, when devising a partial pre-computation strategy. For the point queries, the processing overhead is an important consideration. Our study shows that without careful coordination between pre-computation and query processing, it would be difficult to realize any significant gain in query performance from the pre-computation. We propose a *Cover-based Query Processing* method, which presumes that the pre-computed aggregates form a cube, i.e., PC Cube. The cover-based query processing method is low in processing overhead. We

also prove that our method is able to find one of smallest sets of cells that must be accessed in order to answer a given point query.

Member Selection Algorithm (MSA)–Greedy is proposed to construct a PC Cube. MSA-Greedy is efficient, and its time complexity is $O(k*l*n)$, where k is the number of dimensions, l is the number of members selected and n is the average number of members in each dimension. Our experiments show that the selection by MSA-Greedy is also effective because the PC Cube generated by the MSA-Greedy leads to much shorter time required to answer point queries than a randomly selected PC Cube. It also has the best overall performance among variations of member selection algorithms.

6.2 Future Work

This thesis presents a new approach for the problem of partial pre-computation that is optimised for the workload of point queries. There are some interesting aspects with this approach that deserve more in-depth investigations. Some of them are listed here:

- In this thesis, we propose a Greedy Member Selection algorithm and some of its variations. It is stili interesting to find some other member selection algorithms, like the algorithm that may not always include all primary members.
- Our query processing method is proposed for handling point queries. It can be extended to process the region queries. The query decomposition for region queries can be performed in a way similar to that for point queries. But a simple extension may not be adequate. It will be interesting to study the optimisation problem involved in processing the region queries.

BIBLIOGRAPHY

[AAD+96] Sameet Agarwal, Rakesh Agarwal, Prasad M. Deshpandre, Asnish Gupta, Jeffrey F.Naughton, Ragnu Ramakrishnan, Sunita Sarawagi. On the Computation of Multidimensional Aggregates. In *Proceedings of the 22nd VLDB Conference*, Bombay, India, pages 506-521, 1996.

[AGS97] R. Agrawal, A. Gupta, S. Sarawagi. Modelling Multidimensional Databases, In *13th International Conference on Data Engineering, (ICDE'97)*, Birmingham, U. K., pages 232-243, April 1997.

[BK99] P. A. Boncz and M. L. Kersten. MIL Primitives for Querying a Fragmented World, *The VLDB Journal*, Springer Verlag, 1999

[BPT97] Elena Baralis, Stefano Paraboschi, and Ernest Teniente. Materialized views selection in a multidimensional database. In *Procceedings of 23rd VLDB Conference*, Athens, pages 156-165, 1997.

[CD97] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. In *ACM SIGMOD Record*, 26(1), March 1997

[CKPS95] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing Queries with Materialized Views, In *Proceedings of the 11th IEEE International. Conference on Data Engineering*, pages 190–200, 1995.

[CL98] J. Chang and S. Lee. Query Reformulation Using Materialized Views in Data Warehouse Environment, In *Proceedings of the 1st ACM International Workshop on Data Warehousing and OLAP*, pages 54–59, 1998.

[DANR96] P.M. Deshpande, S. Agarwal, J.F. Naughton, and R. Ramakrishnan. Computation of Multidimensional Aggregates. Technical Report 1314, University of Wisconsin - Madison, 1996.

[DRT99] A. Datta, K. Ramamritham, and H. Thomas. Curio: A Novel Solution for Efficient Storage and Indexing in Data Warehouses, In *Proceedings of the 25th VLDB Conference*, Edinburgh, Scotland, 1999.

[FPSS96a] U. Fayad, G. Piatetsky-Shapiro, and P. Smyth. The KDD Process for Extracting Useful Knowledge from Volumes of Data. In *Communication of ACM*, 39(11). pages 27-34, 1996

[GBLP96] J. Gray, A. Bosworth, A. Layman, H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tabs, and Sub-Totals. In *Proceedings of International Conference on Data Engineering (ICDE'96)*, New Orleans, February 1996

[GHRU97] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index Selection for OLAP. In *Proceedings of 13th International Conference on Data Engineering (ICDE'97)*, UK, pages 208-219, April 1997

[GM99] H. Gupta and I.S. Mumick. Selection of Views to Materialize Under a Maintenance Time Constraint, In *Proceedings of International Conference on Database Theory (ICDT'99)*, Jerusalem, Israel, January 1999.

[Goil99] Sanjay Goil High Performance On Line Analytic and Data Mining. Ph.D. thesis, Electrical and Computer Engineering, Northwestern University, 1999.

[Hel98] J. Hellerstein. Data Warehousing, Decision Support & OLAP http://redbook.cs.berkeley.edu/lec28.html

[HCKL00] E. Hung, D. Cheung, B. Kao and Y. Liang. An optimization problem in data cube system design. Knowledge Discovery and Data Mining, Lecture Notes in Artificial Intelligence 1805, 74-85. 2000.

[HRU96] V. Harinarayan, A. Rajaraman, and J. D.Ullman. Implementing Data Cubes Efficiently. In *Proceedings of SIGMOD*, pages 205-216, 1996.

[Inm92] W. H. Inmon. Building the Data Warehouse. Wiley, New York, 1992

[KR99] Y. Kotidies and N. Roussopoulos. DynaMat: A Dynamic View Management System for Data Warehouses. In *Proceedings of ACM SIGMOD*, Philadelphia, 1999.

[LRS99] Li, J., Rotem, D., Srivastava, J. Aggregation Algorithm for Very Large Compressed Data Warehouses. In Preceedings of 25th Very Large Database (VLDB) Conference. Edinburgh, Scotland, 1999.

[Luk01] Woshun Luk. ADODA: A Desktop Online Data Analyzer. In 7th International Conference on Database Systems for Advanced Applications (DASFAA'01), Hong Kong, China, April 2001.

[LW96] C. Li and X.S. Wang. A Data Model for Supporting On-line Analytical Processing. In *Proceeding Conference on Information and Knowledge Management*, November, pages 81-88, 1996.

[PKL02] C.S. Park, M.H. Kim and Y.J. Lee. Finding an Efficient Rewriting of OLAP Queries Using Materialized Views in Data Warehouses. *Decision Support Systems*, Vol.32, No.4, pages 379-399, 2002.

[Olap97] OLAP Council, The OLAP glossary. http://www.olapcouncil.org. The OLAP Council, 1997

[PS99] P. Vassiliadis and T. Sellis. A Survey of Logical Models for OLAP Databases. In *ACM SIGMOD Record*, Vol. 28, No. 4, 1999.

[RS97] K.A. Ross and D. Srivastava. Fast Computation of Sparse Datacubes. In *Proceedings of the 23rd VLDB Conference*. pages116-125, 1997.

[SAG96] S. Sarawagi, R. Agrawal, A. Gupta. On Computing the Data Cube. *Research Report* 10026, IBM Almaden Research Center, San Jose, California, 1996.

[Sap99] C. Sapia. On Modeling and Predicting User Behavior in OLAP Systems, *In Proceedings of the International CAiSE Workshop on the Design and Management of Data Warehouses (DMDW99)*, Heidelberg June 1999.

[SDJL96] D. Srivastava, S. Dar, H. V. Jagadish, A. Y. Levy. Answering Queries with Aggregation Using Views. In Proceedings of 22nd VLDB Conference, Bombay, India, pages 318-329, 1996

[SDN98] A. Shukla, P. Deshpande, and J. Naughton. Materialized View Selection for Multidimensional Datasets. In *Proceeding of the 24th VLDB Conference*, New York, 1998.

[SDN00] Shukla A., Deshpande P., Naughton J.F., Materialized View Selection for Multi-Cube Data Models. In *Proceedings of EDBT*. pages 269-284, 2000.

[SM94] Sunita Sarawagi and Michael Stonebraker. Efficient Organization of Large Multidimensional Arrays. In *Proceedings of International Conference on Data Engineering (ICDE'94)*. Pages 328-336, 1994.

[SS99] Souza, M.F.,Sampaio, M.C. Efficient Materialization and Use of Views in Data Warehouses. In *ACM SIGMOD Record*. Vol.28, No.1, 1999.

[SWS02] Kurt Stockinger, Kesheng Wu and Arie Shoshani. Strategies for Processing ad hoc Queries on Large Data Warehouses. In *Proceedings of DOLAP'02* VA, USA November, 2002

[Vai98] Aejandro A. Vaisman. Data Warehousing, OLAP, and Materialized Views: a Survey Technical Report TR015-98, University of Buenos Aires, Computer Science Department, 1998

[Vas98] P. Vassiliadis. Modeling Multidimensional Databases, Cubes and Cube Operations. In *Proceeding of 10th International Conference on Statistical and Scientific Database Management (SSDBM)*, Capri, Italy, 1998

[YL99] Yu, J.X., Hongjun Lu. Hash in Place with Memory Shifting: Datacube Computation Revisited. In *Proceedings of 15th International Conference on Data Engineering*. page: 254 March 1999

[ZDN97] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An Array-based Algorithm for Simultaneous Multidimensional Aggregates. In *Proceedings of ACM SIGMOD*. pages 159-170, 1997.

[Zhu98] H. Zhu. On-Line Analytical Mining of Association Rules. M.Sc. thesis, Computing Science, Simon Fraser University, 1998.

[ZYY01] C. Zhang, X. Yao and J. Yang. An Evolutionary Approach to Materialized Views Selection in a Data Warehouse Environment. *IEEE Transactions on Systems, Man and Cybernetics, Part C*, 31(3):282-294, August 2001