ABSTRACT

ASGHARZADEH TALEBI, ZOHREH. Exact and Inexact Methods for Solving the View and Index Selection Problem for OLAP Performance Improvement. (Under the direction of Dr. Yahya Fathi and Dr. Rada Chirkova).

In on-line analytical processing (OLAP), precomputing (*materializing as views*) and *indexing* auxiliary data aggregations is a common way of reducing query-evaluation time (cost) for important data-analysis queries. We consider an OLAP view- and index-selection problem as an optimization problem, where (i) *the input* includes the data-warehouse schema, a set of data-analysis queries of interest, and a storage-limit constraint, and (ii) *the output* is a set of views and indexes that minimizes the total cost of evaluating the input queries, subject to the storage limit. While greedy and other heuristic strategies for choosing views or indexes might have some success in reducing the cost, it is highly nontrivial to arrive at a globally optimal solution, one that reduces the processing cost of typical OLAP queries as much as is theoretically possible.

In this dissertation we present a systematic study of the OLAP view- and index-selection problem. Our specific contributions are: (1) we introduce an integer programming model for OLAP view- and index-selection problem; (2) we develop an algorithm that effectively and efficiently prunes the space of potentially beneficial views and indexes of the problem, and provide formal proofs that our pruning algorithm keeps at least one globally optimal solution in the search space, thus the resulting integer-programming model is guaranteed to find an optimal solution; this allows us to solve realistic-size instances of the problem within reasonable execution time. (3) we develop a family of algorithms to further reduce the size of the search space so that we are able to solve larger instances of the problem, although we no longer guarantee global optimality of the resulting solution; and (4) we present an experimental comparison of our proposed approach with other approaches discussed in the open literature. Our experiments show that our proposed approach to view and index selection results in high-quality solutions — in fact, in the *global optimal* solu-

tions for many realistic-size problem instances. Thus, it compares favorably with the well-known OLAP-centered approach of [13] and provides for a winning combination with the end-to-end framework of [2] for generic view and index selection.

Exact and Inexact Methods for Solving the View and Index Selection Problem for OLAP Performance Improvement

by

Zohreh Asgharzadeh Talebi

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fullfillment of the
requirements for the Degree of
Doctor of Philosophy

Operations Research

Raleigh, North Carolina

2010

APPROVED BY:

Dr. Yahya Fathi Co-Chair of Advisory Committee	Dr. Rada Chirkova Co-Chair of Advisory Committee
Dr. Matthias Stallmann	Dr. Carla Savage

DEDICATION

To My Parents...

BIOGRAPHY

Zohreh Asgharzadeh Talebi has received her B.S. degree in Industrial Engineering from Sharif University of Technology in 2003 and her M.S. degree in Operations Research from North Carolina State University in 2006.

While studying at North Carolina State University, Zohreh has been an active member of several student associations. She served as a president of Omega Rho Honor Society NCSU chapter, and as a secretary of INFORMS NCSU chapter. In 2006 Zohreh won the Elmaghraby distinguished student award. She is a member of Phi Kappa Phi, Omega Rho, INFORMS, SIAM, and IEE. Currently, Zohreh is working as an operations research analyst at SAS Institute in Cary, NC.

ACKNOWLEDGMENTS

I would like to offer my sincerest gratitude to my advisors Dr. Yahya Fathi and Dr. Rada Chirkova for their supervision, advice, and their continued support and encouragement. Their invaluable advice toward being a professional is one of my greatest assets which I will carry throughout my life.

I also would like to thank Dr. Matt Stallmann and Dr. Carla Savage for serving on my committee. I have benefited a lot from Dr. Stallmann's comments on my work. Also Dr. Savage's Graph Theory course was the most joyful course I have ever taken which affected my dissertation path significantly.

For the last year and a half I was lucky to have Reza beside me who has been a great source of motivation and joy in my life. I would like to thank him very much.

Last and most, I would like to express my deepest affection for my parents, who I dedicate this work to, and my brothers and sister who are always there for me. It has not been easy living so far away from them, but their motivation and support made this journey a lot easier.

TABLE OF CONTENTS

List of Tables . . . . . **vii**

List of Figures . . . . . **xi**

Chapter 1 Introduction **1**
1.1 Contributions 4
1.2 Dissertation Structure 6
1.3 Related Work 7

Chapter 2 Preliminaries **10**
2.1 Cost Model 12
2.2 Problem Statement 17

Chapter 3 The Exact Approach **19**
3.1 Integer Programming Model IP1 19
3.2 Reducing the Set of Views 23
3.3 Reducing the Set of Indexes 25

Chapter 4 Experiments with Our Exact Approach **33**
4.1 Constructing Instances 34
4.2 Reducing the Search Space 35
4.3 Scalability of the Exact Model IP2 37
4.4 Summary of Observations 38

Chapter 5 Inexact Methods **39**
5.1 Model IPN 40
5.2 Model IPNIRp 44
5.3 Model IPVs 45

Chapter 6 Experiments with Our Inexact Methods **47**
6.1 Experiments with the Inexact Model IPN 48
6.2 Experiments with the Inexact Model IPNIRp 55
6.3 Experiments with the Inexact Model IPVs 58
6.4 Comparison with the Heuristic Approach ACN 64
6.5 Comparing with the Heuristic Approach GHRU 70
6.6 Summary of Observations 75

Chapter 7 Conclusions and Future Research **77**
7.1 Future Research Avenues 78

References **80**

Appendices **76**

Appendix A An Algorithm to Build Digraph G_v for View v **85**
Appendix B An Algorithm to Find the Elements of the Set $\Pi'(v)$ for view v **88**
Appendix C An Example for Constructing the Set $\Pi''(v)$ for a Given View v **90**

LIST OF TABLES

Table 2.1 The Sales table of Example 3 14

Table 2.2 The Company table of Example 3 14

Table 2.3 The Time table of Example 3 14

Table 2.4 The table for view V of Example 3 15

Table 2.5 The table for the raw-data view of Example 3. Since the grouping attributes of the raw-data view is all of the dimension attributes of the stored data, the columns related to aggregations (i.e. Sum(QtySold) and Max(QtySold)) are the same as the column QtySold. 15

Table 2.6 The table for view `V1` of Example 4 17

Table 4.1 Comparison of the number of views and indexes in models *IP1* and *IP2* for the instances over the 7-attribute TPC-H database. 36

Table 4.2 Comparison of the number of views and indexes in models *IP1* and *IP2* for the instances over the 13-attribute TPC-H database. 36

Table 4.3 The time required to solve the instances over the 7-attribute database using model IP2, and the time required to build the search space of views and indexes. 37

Table 4.4 The time required to solve the instances over the 13-attribute database using model IP2, and the time required to build the search space of views and

indexes. We could not solve the last four instances as the computer ran out of memory. 38

Table 6.1 The value of cost obtained using model IPN and the optimal value of cost (obtained from solving model IP2) for instances over the 13-attribute database. α is defined in the context of Formula 4.1 for storage space. 49

Table 6.2 The value of cost obtained by model IPN and the optimal value of cost (obtained from solving model IP2) for instances on the 13-attribute database. Queries in instances 1 and 2 are from the upper levels of the view lattice, queries in instances 3-7 are from the middle levels of the view lattice, and queries in instances 8-12 are from the lower levels of the view lattice. 50

Table 6.3 Comparison of (1) the value of cost obtained from solving model IPN with the value of cost obtained from solving model IP2 (optimal), (2) the total time required to solve each of the instances using model IPN with the total time required to solve the same instance using model IP2, and (3) the number of indexes in the search space of indexes of model IPN with the corresponding number in model IP2 for each of the instances over the 7-attribute database. 52

Table 6.4 Comparison of (1) the value of cost obtained from solving model IPN with the value of cost obtained from solving model IP2 (optimal), (2) the total time required to solve each of the instances using model IPN with the total time required to solve the same instance using model IP2, and (3) the number of indexes in the search space of indexes of model IPN with the corresponding number in model IP2 for each of the instances over the 13-attribute database. 53

Table 6.5 Comparison of the value of cost obtained from model IPNIRp with the value of cost obtained from model IPN for different values of p for a large instance on the 13-attribute database with 100 queries. 55

Table 6.6 The value of $p_{0.01}$ for instances on the 13-attribute database. 56

Table 6.7 The value of $p_{0.01}$ for instances on the 17-attribute database. 57

Table 6.8 Solving a large instance with 70 queries on the 17-attribute database using model IPNIRp with p=20. We could not solve this instance using p=10 in one hour. 58

Table 6.9 The result of solving instance 1 over the 17-attribute database using model IPVs for different values of s. 59

Table 6.10 The result of solving instance 2 over the 17-attribute database using model IPVs for different values of s. 59

Table 6.11 The result of solving instance 3 over the 17-attribute database using model IPVs for different values of s. 60

Table 6.12 The result of solving instance 4 over the 17-attribute database using model IPVs for different values of s. 60

Table 6.13 Comparing model IPVs when $s = 1$ and model IPN for 18 instances over a 13-attribute database. Instances are the same as instances in Table 6.6. 61

Table 6.14 Comparing model IPVs when $s = 1$ and model IPNIRp with $p = p_{0.01}$ for 18 instances over a 13-attribute database. Instances are the same as instances in Table 6.6. 62

Table 6.15 Comparing model IPVs when $s = 1$ and model IPN for some of the instances in Table 6.13 when $\alpha = 0.1$. 63

Table 6.16 Comparing model IPVs when $s = 1$ and model IPN for some of the instances in Table 6.13 when $\alpha = 0.2$. 63

Table 6.17 Comparison of (1) the costs obtained from solving models IPN and IPACN, and (2) the time required to solve each of these instances using models IPN and IPACN. Instances are over the 17-attribute database. Each instance has 20 queries. 66

Table 6.18 Comparison of (1) the costs obtained from solving models IPN and IPACN, and (2) the time required to solve each of these instances using models IPN and IPACN. Instances are over the 13-attribute database. Each instance has 20 queries. 68

Table 6.19 The value of cost obtained from GHRU when it is applied on different search spaces for 6 instances over the 7-attribute database. 72

Table 6.20 The total execution time of applying GHRU on different search spaces for 6 instances over the 7-attribute database. (In these instances, the time required to find each search space is significantly shorter than the time required to apply GHRU on that search space.) 73

Table 6.21 Comparison of (1) the cost obtained from solving model IPN with the cost obtained by algorithm GHRU, and (2) the execution times of solving IPN with the required times to apply algorithm GHRU for each of the instances over a 7-attribute database. Each instance has 10 queries. 73

LIST OF FIGURES

Figure 3.1 Digraph G_v for Example 5. 28

Figure 6.1 Comparison of the size of the search space of views and indexes in models IPN and IPACN for the instances over the 17-attribute database. . . . 67

Figure 6.2 Comparison of the size of the search space of views and indexes in model IPN and IPACN for the instances over the 13-attribute database. . . . 69

Chapter 1

Introduction

On-line analytical processing (OLAP) and data warehousing are specially designed to enable executives, managers, and analysts to make better and faster decisions. OLAP applications include marketing, business and management reporting, budgeting, forecasting, health care, systems analysis, etc. Users are mainly interested in summary information of a measure as a function of some business aspects (dimensions). For instance, consider a warehouse that keeps the information related to a company's sales. For each sales event, interesting dimensional information can be *the product sold*, *the time of sale*, and *the customer*.

In practice, the number of dimensions for a data warehouse can be relatively large, and each dimension can have a number of distinct *attributes* which are stored in a separate *dimension table* (e.g., attributes of "product" could be its "color", its "size", its "weight", etc.). User queries typically specify these attributes, and preparing a response to a query could involve an extensive search through a number of dimension tables for proper attribute values. As a result, it may be quite time consuming to answer aggregate queries directly from the stored data in the database. In order to accelerate query evaluation, a common practice is to pre-compute and store (materialize) auxiliary data such as views and indexes (see [10]).

As we shall explain later, for a given database, the total number of distinct views and indexes can be extremely large; hence it is not always practicable to materialize

all potentially beneficial views and indexes due to the limited amount of storage space that we can physically maintain. This is where the problem of selecting a subset of views and indexes arises. In this context we are typically interested in making the selection that would maximize the associated benefit (e.g., minimize the response time for a given collection of queries) while observing the storage space limit.

Throughout this work we assume that data warehouses under consideration have the star schema [10] with a single fact table and several dimension tables under the realistic assumption that in each table all rows have a single fixed length (upper bound), and that the time (cost) of evaluating a query is proportional to the number of stored-data tuples scanned by the query-processing system when evaluating the query [13, 14]. We now give an example that shows how materialized views and indexes may speed up the evaluation times of aggregate queries.

Example 1. Consider a data warehouse with three stored tables: *Sales(CID,DateID, QtySold), Company(CID,CompName,State), and Time(DateID,Day,Month,Year).* Here, *Sales* is the fact table in the star schema of the data warehouse, and *Company* and *Time* are dimension tables, with key attributes underlined.

Suppose the query workload has two queries, *Q1* and *Q2*. *Q1* asks for the total quantity of products sold in November 2006. *Q2* asks for the maximum product quantity sold in 2006 to companies in North Carolina. The SQL representation of the queries is as follows:

Q1:

```
SELECT SUM(QtySold)
FROM Sales s, Time t, Company c
WHERE s.DateID = t.DateID
AND s.CID = c.CID AND Year = 2006
AND Month = 'Nov';
```

Q2:

```
SELECT MAX(QtySold)
 FROM Sales s, Time t, Company c
 WHERE s.DateID = t.DateID
 AND s.CID = c.CID AND State = 'NC'
AND Year = 2006;
```

Each of *Q1* and *Q2* can be answered using either the original stored fact and

dimension tables or the raw-data view[1] of this data warehouse. (Under the realistic assumption that the warehouse contains indexes on the primary keys of all the stored tables, these two strategies for evaluating *Q1* have similar costs; the same point holds about *Q2*.)

We can use techniques from [1, 14] to show that the following view V can be used to give exact answers to each of *Q1* and *Q2*.

V:
SELECT s.CID, Year, Month, State, SUM(QtySold) AS SumQS, MAX(QtySold) AS MaxQS
FROM Sales s, Time t, Company c
WHERE s.DateID = t.DateID AND s.CID = c.CID
GROUP BY s.CID, Year, Month, State;

When we materialize V as a table in the data warehouse, under a wide range of realistic assumptions on the contents of the stored data, there is an evaluation-time benefit to use V instead of the raw-data view (or of the original stored tables) in answering each of *Q1* and *Q2*. This benefit is present even when we assume that no indexes have been created on the stored table *V*. However, if we create a B+-tree index I on the sequence of attributes *Month, State, Year, CID* of the table *V*, we may be able to further cut the evaluation costs for *Q1* using *V*. The reason for this further cost reduction is that index I permits the OLAP system to scan only those tuples of the view that contribute directly to the answer to *Q1*. (These are the tuples that have the sales information only for November.) At the same time, in answering query *Q2* using view *V*, index I is not beneficial, because the order of the attributes in I is in such a way that I cannot be used to limit the number of tuples scanned for *Q2*. □

[1]The *raw-data view* of a star-schema data warehouse is the table resulting from the star join of all the stored (both fact and dimension) tables. The raw-data view can be defined in SQL as a *GROUP BY*, on all the dimension attributes of the stored data, of the table resulting from the star join.

The prominent role of materialized views and indexes in improving query-processing performance has long been recognized, see, for instance, [7, 22]. Enterprise-class database-management systems that provide modules for *generic* view and index selection include Microsoft SQL Server [2, 23] and DB2 [7].

We consider the following optimization problem that we refer to as the *OLAP view- and index-selection problem*. Given a data warehouse schema, a set of data-analysis queries of interest, and an upper bound b on the available storage space, find a collection of views and indexes that would fit within the storage limit b and would minimize the cost measure (evaluation time) for the given queries.

Since the total number of possible views (subsets of the set of attributes) and their associated indexes (permutations of the attributes in the given view) is finite, in theory this problem can always be solved using a complete enumeration of all solutions. But even for a database with a relatively small number of attributes, this approach is not practicable since the total number of such solutions can be extremely large. Note that for a collection of k attributes, we have 2^k views in the data cube and 2^{2^k} distinct subsets of views, and for each view v we have $(|v|)!$ possible indexes, where $|v|$ represents the number of attributes in view v. In fact, NP-completeness of a variant of the problem described here is proved in [13]. Thus, it is natural to look for heuristic solutions. Well-known past efforts in this direction include [2, 13]; we discuss these approaches in detail in Section 1.3.

1.1 Contributions

Our approach for solving the view- and index-selection problem is to develop an integer programming model for this problem and to use the structural properties of the views and indexes to reduce the number of potential views and indexes in the search space of this model. More specifically, our contributions are as follows:

1- We model the view- and index-selection problem as an integer programming (IP) model.

2- We develop an algorithm that effectively and efficiently prunes the search space of potentially beneficial views and indexes (Chapter 3). The pruned search space significantly reduces the size of our IP model, so that for realistic-size instances of the problem this IP model can be solved efficiently by an IP solver such as CPLEX [15].

3- We provide formal proofs that our pruning algorithm keeps in the search space at least one globally optimal solution. Thus, the solution obtained after solving the corresponding IP model is guaranteed to be globally optimal. This includes many problem instances of practical interest.

4- We develop a family of algorithms to further reduce the size of the search space. In this reduction, we only keep a collection of promising views and indexes and remove many other feasible solutions. With this reduction, we bring the size of the search space down to a manageable level even for larger instances of the problem. However, we can no longer guarantee that the solution obtained by the IP model is optimal for the original problem. Thus, our proposed algorithms in this context are IP-based *inexact* methods (*heuristic procedures*) for solving the OLAP view- and index-selection problem.

5- We present an experimental comparison of our IP-based inexact approach with an OLAP-modified view- and index-pruning approach of [2]. Our experiments show that for the special case of OLAP queries, the resulting IP model is small enough to solve by an IP-solver such as CPLEX within reasonable execution time while still retaining solutions that have significantly better (i.e., lower) cost than those produced when the OLAP-modified approach of [2] is used as a heuristic to reduce the size of the search space.

6- We report the results of a computational experiment where we evaluate the performance of the well-known view- and index-selection approach of [13], which we refer to it as Algorithm GHRU, when it is applied on several different search spaces that we propose in Chapters 3 and 5. In our experiments we observed

that when we apply GHRU on the original search space of views and indexes, we obtain solutions which are far away from the optimal solutions. Furthermore, when we reduce the search space of views and indexes through our proposed approach (prior to applying GHRU), then GHRU selects a better combination of views and indexes as compared with those it originally obtains.

In Chapter 6 we report the results of applying our IP-based heuristic and the approach of [13] on a collection of instances.

Our research in the context of selecting views and indexes has resulted in several publications including [3], [4], and [21].

1.2 Dissertation Structure

In the remainder of this chapter, we review the related work. In Chapter 2 we present a formal definition of the view- and index-selection problem and discuss the formulation and settings. In Chapter 3 we present the integer programming model for view- and index-selection problem. In this chapter we also propose specific procedures to reduce the size of the initial search space of views and indexes, with the goal of reducing the size of the integer programming model, while maintaining that the resulting optimal solution of the IP model is also globally optimal for the original problem. In Chapter 4 we present our experimental results for the proposed exact approach of Chapter 3. In Chapter 5 we propose methods for further reducing the size of the search space of views and indexes in order to reduce the size of the resulting IP model, but we no longer guarantee global optimality. We present and discuss our experimental results for the proposed inexact methods in Chapter 6. In Chapter 7 we conclude our work.

1.3 Related Work

The prominent role of materialized views and indexes in improving query-processing performance has long been recognized, see, for instance, [7] and [22]. Enterprise-class database-management systems that provide modules for *generic* view and index selection include Microsoft SQL Server (see [2] and [23]) and DB2 (see [5], [26], and [28]). At the same time, while it can be relatively easy to *improve to some degree* query-evaluation costs by using, for instance, greedy strategies for choosing indexes or views, it is highly nontrivial to arrive at a *globally optimal solution,* i.e., one that reduces the processing costs of the given OLAP queries as much as is theoretically possible. As observed in [17] and to the best of our knowledge, there is no known approximation algorithm (with nontrivial performance guarantees) for the view- and index-selection problem in the open literature. Hence it is natural to look for heuristic approaches for solving the problem. Well-known past efforts in this direction include the work by Agrawal et al. in [2] and Gupta et al. in [13].

Two families of algorithms for solving the problem of view and index selection in a generalization of the OLAP setting is proposed in [13]. Karloff and Mihail in [17] disproved the strong performance bounds of these algorithms, by showing that the underlying approach of [14] cannot provide the stated worst-case performance ratios unless P=NP.

A well-known tool for automated selection of materialized views and indexes for a wide variety of query, view, and index classes in relational database systems is presented in [2]. The approach of [2], implemented in Microsoft SQL Server, is based partly on the authors' previous work (see [11]) on index selection. The contributions stated in [2] are (i) an end-to-end framework for view and index selection in practical systems, and (ii) a module for building (pruning) the search space of potential views and indexes for a given query workload. In this dissertation, we experimentally show that our proposed pruning algorithms for view and index selection fare well when compared (in the special case of OLAP queries) to the pruning algorithm proposed by [2]. This implies that our proposed algorithms are also suitable for complementing

the overall framework proposed in [2] in the special case of OLAP.

A uniform approach for selecting views and indexes for OLAP queries is proposed in [12]. This approach considers view- and index-maintenance costs alongside query-response costs. The paper proposes to use a "bond energy" algorithm for initial clustering of indexes, and then to apply a partitioning method to select a set of views or indexes. Once the best partition is found, views or indexes are eliminated in a greedy manner, until the storage-space constraint is satisfied. This paper leaves out most implementation details as well as any performance study of the proposed approach, which makes the approach rather hard to compare with other work.

Other past work considers either selection of views only (VSP) (see, e.g., [6, 16, 24, 27] and references therein) or selection of indexes only (ISP) (see, e.g., [8, 9, 19, 11] and references therein) for OLAP. Our work in this dissertation differs from the work of these papers in that we consider the problem of selection of views and indexes simultaneously. Yang and colleagues [27] propose an integer programming model for selecting the search space of views, coupled with a heuristic algorithm for selecting views from the resulting space. They include both query-processing costs and view-maintenance costs in the cost measure. Note that in our approach we use integer programming at the stage of view selection, rather than at the stage of forming or pruning the search space, and that our search space includes not only views but also indexes.

Caprara et al. in [8] introduced an uncapacitated facility location model in the context of integer programming for solving ISP under the storage space constraint to minimize the query response time and maintenance cost of indexes in a database. They proposed optimal and heuristic methods to solve their model. They also proposed an algorithm to solve very large scale instances of ISP.

Approaches proposed in [19] and [11] for ISP are configuration based where each configuration is composed of several indexes. In [19] a genetic algorithm is developed for solving ISP to select some predefined configurations that maximize the total benefit of configurations minus the maintenance cost of the indexes in configurations. In the setting of the approach in [19], each query can be responded using

indexes of at most one configuration. In [11], indexes in a configuration are built based on the "useful" columns of the queries in the workload, and k single-column indexes are selected using a Greedy(m,k) algorithm where first the combination of m best indexes is selected by enumeration and the next $k - m$ indexes are selected in a greedy manner. After selecting single-column indexes, the approach of [11] considers multi-column indexes as well.

Chaudhuri et al. in [9] studied the hardness of ISP by considering two subproblems of ISP: one with the storage space constraint and the other with the clustered index constraint (every table in the database has at most one clustered index). They proved that both of these subproblems are NP-hard. They also proposed a heuristic approach based on a knapsack model [20]. In their approach, they considered the interactions of indexes on queries as well: They assigned a benefit to each index not only by considering the effect of that index on that query, but also by considering the effect of other indexes that can be useful for the same query. In this heuristic, the benefit of each index is the sum of its benefit for all queries in the workload. They use a greedy algorithm to solve the knapsack problem and have instance specific guarantees for their algorithm. The number of indexes that they have to consider for each query can be very large, which makes the benefit assignment to indexes substantially more difficult.

Chapter 2

Preliminaries

In this chapter, first we define the scope of the view- and index-selection problem that we consider, i.e., the type of the database, queries, views, and indexes. Next, we introduce a cost model for this problem. Finally, we provide a formal definition of the view- and index-selection problem.

We consider relational select-project-join queries with grouping and aggregation (SPJGA) in star-schema data warehouses [10, 18]. Similarly to [13, 14, 16, 24], we assume users frequently ask a limited number of SPJGA queries, such as itemized daily sales reports, for a variety of parameters for products, locations, etc. Thus, we assume parameterized queries, by allowing arbitrary constant values in the WHERE clauses of the queries, and assume that specific values of these constants are not known in advance. We consider star-schema data warehouses with a single fact table and several dimension tables, under the following realistic assumptions. First, in each base table all rows have a single fixed (upper bound on) length. Second, the fact table has many more rows than each dimension table. Finally, we assume that each base table has a single index, on the table's key.

Our (original) search space of views is the *view lattice* defined in [14], which includes all star-join views with grouping and aggregation (JGA views) on the base

tables. Each lattice view (1) has grouping on some of the attributes of the database, and (2) has aggregation on *all* the attributes aggregated in the input queries, using all the aggregation functions in the queries (such views are called "multiaggregate views" [1]).

B+-tree indexes play an important role in answering queries efficiently by reducing the number of rows needed to be scanned to answer the queries. As we will explain in Subsection 2.1, the ordering of attributes in an index is important in answering a query using that index. A B+-tree index can be defined by any permutation of any subset of unaggregated (grouping) attributes of a view. In our study we consider only *fat indexes* over the lattice views — an index for a given view v is said to be a *fat* index if it is associated with a permutation of *all* of the grouping attributes of view v.

We say a SPJGA query q can be answered using a JGA view v if and only if the set of grouping attributes of v is a superset of the set of attributes in the `GROUP BY` clause of q and those attributes in the `WHERE` clause of q that are compared with constants. Furthermore, if view v is chosen for answering query q, then at most one index of view v can be used to answer query q. By definition, each query q can be answered using the raw-data view in the view lattice.

Example 2. Consider view $v = \{a, b, c, d\}$ and queries q_1=$\{a, b\}$ and q_2=$\{d, e\}$ where the letters a, b, c, d, and e represent distinct attributes in the database. Attributes a, b, c, and d are the grouping attributes of view v. Attributes a and b form the collection of attributes that are either in the `GROUP BY` clause of q_1, or among those attributes in the `WHERE` clause of q_1 that are compared with constants. Similarly, attributes d and e form the collection of attributes that are either in the `GROUP BY` clause of q_2 or among those attributes in the `WHERE` clause of q_2 that are compared with constants. Since $q_1 \subseteq v$, view v can answer query q_1. However, since $q_2 \nsubseteq v$, view v cannot answer query q_2. □

For ease of presentation, throughout this dissertation we use the letter v to represent both a view and the collection of grouping attributes for that view, and we use

the letter q to represent both a query and the collection of attributes in the `GROUP BY` clause of that query plus those attributes in the `WHERE` clause of the query that are compared with constants. Also, we use the character π to represent both a fat index and the permutation vector associated with that fat index.

2.1 Cost Model

The cost model that we use is similar to the one proposed in [13], i.e., the cost of answering query q using view v is the size of that portion of v that must be processed (scanned) in order to construct the result of query q. We measure the size of a view or a portion of a view as the number of rows in that view or in the portion thereof.

When we answer query q using only view v with no indexes, then we have to scan all rows of v. Hence the corresponding cost is equal to the size of view v itself. However, when we answer query q using view v and an index π of v, we only need to read the part of v referenced by π with respect to q, hence the corresponding cost is potentially smaller. Naturally the cost of answering a query in this situation depends on the actual contents of the data set under consideration, and it can be factually determined only after we have scanned the corresponding data. But in order to compare various courses of action and devise an appropriate action plan we need to evaluate this cost prior to scanning the data. Gupta et al. in [13] propose an approach to obtain a reasonable estimate for this cost using the available information about the size of various views in the view lattice. We adopt this approach to estimate the cost coefficients in our models, and in the remainder of this section we introduce and explain this approach.

Suppose A_1 is the set of attributes in the `GROUP BY` clause of query q, and A_2 is the set of attributes that are compared with constants in the `WHERE` clause of query q. Also suppose B is the set of grouping attributes of view v. Let π represent an index over view v, i.e., a permutation of attributes in the set B. (Recall that we use the character π to represent both the index and the corresponding permutation vector of

its attributes). As mentioned earlier, view v can be used to answer query q if and only if $(A_1 \cup A_2) \subseteq B$.

Let us use the notation $c_q(v, \pi)$ to denote the estimated cost of answering query q using view v and its index π. Naturally $c_q(v, \pi)$ is defined only if view v can be used to answer query q, i.e., if $(A_1 \cup A_2) \subseteq B$. The value of $c_q(v, \pi)$, however, depends on the size of view v and the relationship between its index π and the collection of attributes A_1 and A_2. In particular, let $v_\pi(q)$ denote the view whose set of grouping attributes is identical to the largest subset of A_2 that forms a prefix (not necessarily proper) of π, i.e., the largest subset of attributes that are compared with constants in the `WHERE` clause of q, and form a prefix of π. Then [13] suggests the following formula to estimate the cost of answering query q using view v and index π

$$c_q(v, \pi) = \frac{size(v)}{size(v_\pi(q))} \tag{2.1}$$

where $size(v)$ represent the size of view v as defined above[1]. For notational convenience, we use $v = \emptyset$ to represent the view which is aggregated on all of the attributes of the database, and define $size(\emptyset) = 1$. Furthermore, following the articles by [14], [21], and [13], we assume that the total cost of answering a given collection of queries is the sum of the costs of evaluating the individual queries.

In order to determine the size of a view in the view lattice, we can use either the sampling method or the analytical method proposed by [13]. For a given view, if we know that its grouping attributes are statistically independent, we can estimate its size analytically from the size of the raw-data view. In this case the size of the view is the number of distinct values of the grouping attributes of the view. Otherwise, we estimate the size of the view by sampling from the raw data.

For ease of presentations, throughout the rest of this dissertation we assume that for every query q in the workload, all of its attributes are in its `WHERE` clause and they are compared with constants, and that its `GROUP BY` clause is empty, i.e., we assume $q = A_2$. Hence from here onward the notation $v_\pi(q)$ in Equation 2.1 denotes

[1]Note that if the set of the grouping attributes of view v_1 is a subset of the set of grouping attributes of view v_2, then we have $size(v_1) \leq size(v_2)$.

Table 2.1: The Sales table of Example 3

CID	DateID	QtySold
1	1	70
1	2	100
1	3	80
1	6	100
1	8	100
1	7	100
1	9	80
2	1	25
2	3	14
2	6	12
2	7	12
2	8	50
3	2	30
3	4	40
3	5	30
3	6	100
3	8	100

Table 2.2: The Company table of Example 3

CID	Name	State
1	NCFL	NC
2	SouthRoad	SC
3	Unolits	NC

Table 2.3: The Time table of Example 3

DateID	Day	Month	Year
1	1	Nov	2006
2	10	Nov	2006
3	20	Nov	2006
4	5	Dec	2006
5	15	Dec	2006
6	5	Jan	2007
7	15	Jan	2007
8	5	Feb	2007
9	15	Feb	2007

the view whose set of grouping attributes are identical to the largest subset of q that forms a prefix of π. (With minor modification, all results that we obtain here are valid without this assumption.)

Example 3. Consider a data warehouse with three stored tables as in Example 1: *Sales(CID,DateID,QtySold)*, *Company(CID,CompName,State)*, and *Time(DateID,Day,Month,Year)*. Here, *Sales* is the fact table, and *Company* and *Time* are dimension tables. Tables 2.1, 2.2, and 2.3 present (for the purposes of this toy example) the data in these tables.

Suppose the query workload consists of two queries, *Q1* and *Q2*, described in Example 1. *Q1* asks for the total quantity of products sold in November 2006. *Q2* asks for the maximum product quantity sold in 2006 to companies in North Carolina.

As we said in Example 1 we can use techniques from [1, 14] to show that the following view V can be used to give exact answers to each of *Q1* and *Q2*.

Table 2.4: The table for view V of Example 3

CID	Year	Month	State	Sum(QtySold)	Max(QtySold)
1	2006	Nov	NC	250	100
1	2007	Jan	NC	200	100
1	2007	Feb	NC	180	100
2	2006	Nov	SC	39	25
2	2007	Jan	SC	24	12
2	2007	Feb	SC	50	50
3	2006	Nov	NC	30	30
3	2006	Dec	NC	70	40
3	2007	Jan	NC	100	100
3	2007	Feb	NC	100	100

Table 2.5: The table for the raw-data view of Example 3. Since the grouping attributes of the raw-data view is all of the dimension attributes of the stored data, the columns related to aggregations (i.e. Sum(QtySold) and Max(QtySold)) are the same as the column QtySold.

CID	Name	State	DateID	Day	Month	Year	QtySold
1	NCFL	NC	1	1	Nov	2006	70
1	NCFL	NC	2	10	Nov	2006	100
1	NCFL	NC	3	20	Nov	2006	80
1	NCFL	NC	6	5	Jan	2007	100
1	NCFL	NC	8	5	Feb	2007	100
1	NCFL	NC	7	15	Jan	2007	100
1	NCFL	NC	9	15	Feb	2007	80
2	SouthRoad	SC	1	1	Nov	2006	25
2	SouthRoad	SC	3	20	Nov	2006	14
2	SouthRoad	SC	6	5	Jan	2007	12
2	SouthRoad	SC	7	15	Jan	2007	12
2	SouthRoad	SC	8	5	Feb	2007	50
3	Unolits	NC	2	10	Nov	2006	30
3	Unolits	NC	4	5	Dec	2006	40
3	Unolits	NC	5	15	Dec	2006	30
3	Unolits	NC	6	5	Jan	2007	100
3	Unolits	NC	8	5	Feb	2007	100

Q1:

```
SELECT SUM(QtySold)
FROM Sales s, Time t, Company c
WHERE s.DateID = t.DateID
AND s.CID = c.CID AND Year = 2006
AND Month = 'Nov';
```

Q2:

```
SELECT MAX(QtySold)
 FROM Sales s, Time t, Company c
WHERE s.DateID = t.DateID
AND s.CID = c.CID AND State = 'NC'
AND Year = 2006;
```

V:

```
SELECT s.CID, Year, Month, State, SUM(QtySold) SumQS, MAX(QtySold) MaxQS
FROM Sales s, Time t, Company c
WHERE s.DateID = t.DateID AND s.CID = c.CID
GROUP BY s.CID, Year, Month, State
```

Table 2.4 presents the data in this view when it is materialized on the database presented in Tables 2.1, 2.2, and 2.3. Also, both queries *Q1* and *Q2* can be answered by the raw-data view of this data warehouse. Table 2.5 presents the data in the raw-data view.

Now consider the fat index I for view V with the following sequence of attributes:

I=(Month, State, Year, CID)

This index is beneficial to query *Q1* as it has attribute *Month* first which is also an attribute in *Q1*; yet this index is not helpful for *Q2* as the first attribute of index *I*, i.e. *Month*, does not appear in this query. In other words, none of the subsets of the set of attributes of *Q2* forms a prefix of *I*. □

Example 4. For the data warehouse defined in Example 3, suppose view *V1* is defined as follows:

V1:

```
SELECT t.Month, SUM(QtySold),MAX(QtySold)
FROM Sales s, Time t, Company c
```

Table 2.6: The table for view `V1` of Example 4

Month	Sum(QtySold)	Max(QtySold)
Nov	319	100
Dec	70	40

WHERE s.DateID = t.DateID AND s.CID = c.CID
AND Year = 2006
GROUP BY t.Month;

Table 2.6 presents the data in this view when it is materialized on the database presented in Tables 2.1, 2.2, and 2.3. Based on the formula for cost presented in this section, the cost of answering query *Q1* using view V and index I (both defined in Example 3) is: $c_{Q1}(V, I) = \frac{size(V)}{size(V1)} = \frac{10}{2} = 5$ (Note that in this example $v_I(Q1)$=V_1). However, the cost of answering query *Q1* using view V without any of its indexes is $size(V) = 10$, and the cost of answering this query using the raw-data view v_{RD} is $size(v_{RD}) = 17$. □

2.2 Problem Statement

In practical settings, the amount of available storage space is a natural constraint in the (OLAP) view- and index-selection problem, as storing all possibly beneficial views and indexes is infeasible in today's database systems (see [2],[13]). We consider the following OLAP view- and index-selection (*OLAP-VI*) problem: Given a star-schema data warehouse and a set (workload) of parameterized SPJGA queries, our goal is to minimize the estimated evaluation cost of the queries in the workload, by selecting and pre-computing (1) a set of lattice (JGA) views that can be used in answering the queries, and (2) some fat indexes over those views. We consider this minimization problem under a given storage-space limit, which is an upper bound

on the amount of disk space that can be allocated for the materialized views and indexes. Thus, our problem input is of the form $(\mathcal{D}, \mathcal{Q}, b)$, where $\mathcal{D}$ is a database, $\mathcal{Q}$ is the workload (which is a set of parameterized queries), and b is the storage limit.

Definition 1. (*Feasibility*) For a problem input $(\mathcal{D}, \mathcal{Q}, b)$, a set of views and indexes $\mathcal{VI}$ is feasible if (1) each query in $\mathcal{Q}$ can be answered using the views in $\mathcal{VI}$, and (2) the set $\mathcal{VI}$ satisfies the storage limit b.

Definition 2. (*Optimality*) For a problem input $(\mathcal{D}, \mathcal{Q}, b)$, an optimal set of views and indexes is a set of views and indexes $\mathcal{VI}^*$ such that (1) $\mathcal{VI}^*$ is feasible for the problem input, and (2) $\mathcal{VI}^*$ minimizes the cost of evaluating $\mathcal{Q}$ on the database $\mathcal{D}v$, among all feasible sets of views and indexes for the problem input. Here, $\mathcal{D}v$ is the database that results from adding to $\mathcal{D}$ the relations for all of the views and indexes in $\mathcal{VI}^*$.

Definition 3. (*OLAP-VI problem*) For a given problem input $(\mathcal{D}, \mathcal{Q}, b)$, the OLAP view- and index-selection (*OLAP-VI*) problem is the problem of finding an optimal set of views and indexes, as defined above.

A solution for a given instance of *OLAP-VI* problem consists of a set of materialized views $\mathcal{V}^*$ (which includes the raw-data view on $\mathcal{D}$ and all additional views that we choose to materialize), a set Π^* of indexes over the views in $\mathcal{V}^*$, and an association between each element of $\mathcal{Q}$ and its corresponding elements of $\mathcal{V}^*$ and Π^*, i.e., which view in $\mathcal{V}^*$ and which index in Π^* (if any) should be used to answer each query in $\mathcal{Q}$.

Following the work of [13] and [14] we assume that the raw-data view (i.e., the top of the view lattice) is always in the solution, although in the context of our proposed models this assumption can be easily removed.

Chapter 3

The Exact Approach

In this chapter we introduce an integer programming (IP) model for the OLAP view- and index-selection problem (OLAP-VI) and study its properties. Subsequently, we use these properties to remove some variables and constraints from this model and obtain a model which is significantly smaller, yet its optimal solution is guaranteed to be optimal for the original *OLAP-VI* problem. This, in turn, allows us to solve larger instances of the problem.

3.1 Integer Programming Model IP1

In this section we propose an integer-programming model for OLAP-VI problem. For a given problem input $(\mathcal{D}, \mathcal{Q}, b)$, we define the following notation:

V : The set of all views in the view lattice
$\Pi(v)$: The set of all indexes of view v, $\forall v \in V$
$Q(v)$: The set of all queries in $\mathcal{Q}$ that can be answered by view v, $\forall v \in V$.

The cardinality of the set V is 2^k, where k is the total number of distinct attributes

in the database. We use the notation v_j to represent the j^{th} view in the set V, for $j = 1$ to 2^k, and use the letter J to represent the corresponding collection of subscripts (i.e., $J = \{1, 2, 3, \ldots, 2^k\}$).

In order to introduce the decision variables for the integer programming model, we need additional notation as follows. Clearly for each view v, the cardinality of the corresponding set of indexes $\Pi(v)$ is equal to the total number of permutations of the elements of v, i.e., $|\Pi(v)| = (|v|)!$. For a given view v_j we denote its l^{th} index by π_{jl}, for $l = 1, \ldots, (|v|)!$, and, for brevity, we denote the collection of all indexes associated with v_j by Π_j. In other words, we use Π_j to denote the set $\Pi(v_j)$, for $j = 1, \ldots, 2^k$. We use the notation q_i (for $i = 1, \ldots, m$) to denote the i^{th} element of the given set of queries $\mathcal{Q}$, i.e., $\mathcal{Q}$=$\{q_1, q_2, \ldots, q_m\}$. For each $i = 1, \ldots, m$, let $V_i = \{v_j \in V : v_j \supseteq q_i\}$ represent the collection of views that can be used to answer query q_i, and let J_i represent the corresponding collection of subscripts, i.e., $J_i = \{j \in J : v_j \in V_i\}$.

We are now prepared to define the decision variables for the integer programming model. The following variables are defined for subscript values $i = 1, 2, \cdots, m$, $j \in J_i$, and $l = 1, 2, \cdots, (|v_j|)!$.

$$s_{ij} = \begin{cases} 1 & \text{If view } v_j \text{ is used to answer query } q_i \text{ with no index} \\ 0 & \text{Otherwise} \end{cases}$$

$$y_{ijl} = \begin{cases} 1 & \text{If view } v_j \text{ and its index } \pi_{jl} \text{ are used to answer query } q_i \\ 0 & \text{Otherwise} \end{cases}$$

The following variables are defined for subscript values $j = 1, 2, \cdots, 2^k$ and $l = 1, 2, \cdots$, $(|v_j|)!$.

$$t_j = \begin{cases} 1 & \text{If view } v_j \text{ is materialized} \\ 0 & \text{Otherwise} \end{cases}$$

$$x_{jl} = \begin{cases} 1 & \text{If index } \pi_{jl} \text{ of view } v_j \text{ is materialized} \\ 0 & \text{Otherwise} \end{cases}$$

Our *OLAP-VI* problem can now be stated as the following integer programming model that we refer to as model *IP1*. In this model we use the notation c_{ijl} to

represent the value $c_{q_i}(v_j, \pi_{jl})$, which is the estimated cost of answering query q_i using view v_j and its index π_{jl} as defined earlier. Correspondingly we use the notation d_{ij} to represent the estimated cost of answering query q_i using view v_j with no index at all. As stated earlier we have $d_{ij} = size(v_j)$.

$$\text{min} \quad \sum_{i=1}^{m} \sum_{j \in J_i} \left[d_{ij} s_{ij} + \sum_{l=1}^{(|v_j|)!} c_{ijl}.y_{ijl} \right] \qquad \textit{IP1}$$

$$\text{subject to} \quad \sum_{j \in J_i} \left[s_{ij} + \sum_{l=1}^{(|v_j|)!} y_{ijl} \right] = 1 \quad \text{for all } i = 1 \text{ to } m \tag{3.1}$$

$$\sum_{j=1}^{2^k} size(v_j) \left[t_j + \sum_{l=0}^{(|v_j|)!} x_{jl} \right] \leq b \tag{3.2}$$

$$x_{jl} \leq t_j \quad \text{for all } j = 1 \text{ to } 2^k, \text{ and } l = 1 \text{ to } (|v_j|)! \tag{3.3}$$

$$s_{ij} \leq t_j \quad \text{for all } i = 1 \text{ to } m \text{ , and } j \in J_i \tag{3.4}$$

$$y_{ijl} \leq x_{jl} \quad \text{for all } i = 1 \text{ to } m, \ j \in J_i, \text{ and } l = 1 \text{ to } (|v_j|)! \tag{3.5}$$

$$t_1 = 1 \tag{3.6}$$

$$t_j, s_{ij}, x_{jl}, y_{ijl} \in \{0, 1\} \qquad \forall i, j, l \tag{3.7}$$

Equation (3.1) states that each query must be answered by exactly one view and either with no index or with exactly one of its indexes. Constraint (3.2) states that the total storage requirement for the selected views and indexes should not exceed the pre-specified limit b. Recall that we measure the size of each view by the number of rows in that view. Also note that the size of each index for a view is the same as the size of the view itself. Correspondingly we state the storage limit b in terms of the number of rows that we can store. Constraint (3.3) states that index π_{jl} for view v_j can be materialized only if the view itself is materialized. Similarly constraint (3.4) states that query q_i can be answered by view v_j (with no index) only if this view is materialized, and constraint (3.5) states that query q_i can be answered by view v_j and its index π_{jl} only if this index is materialized. Finally, constraint (3.6) simply

states that the raw-data view is always selected[1].

In the *OLAP-VI* problem and the resulting integer programming model *IP1* we consider all views in the view lattice that can answer at least one query in the workload and all their corresponding indexes to be in the search space of the problem. For a realistic size instance of the problem the total number of views and indexes in this search space, and hence the size of the corresponding integer programming model *IP1*, can be quite large. Note that for a database with k dimensions (attributes) the total number of views in the view lattice is 2^k and each view v_j has $(|v_j|)!$ indexes. For an instance of the problem with k attributes and m queries this results in at most $\sum_{i=1}^{m}\left[|J_i| + \sum_{j\in J_i}[(|v_j|)!]\right] + \sum_{j=1}^{2^k}[(|v_j|)!] + 2^k$ variables and at most $\sum_{i=1}^{m}\left[|J_i| + \sum_{j\in J_i}[(|v_j|)!]\right] + \sum_{j=1}^{2^k}[(|v_j|)!] + m + 2$ constraints in the integer programming model *IP1*. Hence even for relatively small values of k and m the resulting integer programming model can be quite large, and the corresponding execution time for solving this model can be excessively long even if we use a relatively fast IP solver such as CPLEX 10 (see [15]). This, in turn, limits the applicability of this approach (i.e., using the integer programming model) to only very small instances of the problem.

In the next two sections we characterize various properties of the views and indexes that appear in an optimal solution for this problem. This allows us to identify a relatively small subset of views and indexes that contain at least one set of optimal views and indexes for the problem. This, in turn, allows us to reduce the size of the corresponding integer programming model for a given instance of *OLAP-VI* problem, hence enabling us to solve larger instances of the problem using this approach within reasonable execution times.

[1]In our IP model we denote this raw-data view by t_1.

3.2 Reducing the Set of Views

In this section we identify a relatively small subset of views V' which, along with their corresponding collection of indexes, contain at least one set of optimal views and indexes.

Proposition 1. Given an instance of the OLAP-VI problem with input data $(\mathcal{D}, \mathcal{Q}, b)$, if a view $v \in V$ is not a superset of at least one query in the set $\mathcal{Q}$, then the problem has an optimal solution that does not include view v.

Proof. This result follows from the fact that all view sizes are positive and, since any such view cannot be used to answer any query in $\mathcal{Q}$, then it cannot affect the value of the objective function. □

Lemma 1. If view v_1 is a proper subset of view v_2 and both v_1 and v_2 can answer the same set of queries, i.e., $Q(v_1) = Q(v_2)$, then for any index π_2 of view v_2 there exists an index π_1 of view v_1 such that for every query q in $Q(v_2)$ (or equivalently in $Q(v_1)$) we have $c_q(v_1, \pi_1) \leq c_q(v_2, \pi_2)$.

Proof. Recall that for any query $q \in Q(v_2)$ we have $c_q(v_2, \pi_2) = \frac{size(v_2)}{size(v_{\pi_2}(q))}$, where $v_{\pi_2}(q)$ is the largest subset of q that forms a prefix of π_2. We now construct an index π_1 for v_1 as follows. For each $q \in Q(v_2)$ (or equivalently in $Q(v_1)$), find its corresponding set $v_{\pi_2}(q)$. Each of these sets has the property that it is a prefix of π_2. Let v' be the largest such set. It follows that v' contains all elements of every set $v_{\pi_2}(q)$ for all $q \in Q(v_2)$. Let the elements of v' be the first group of elements in π_1 in the same order that they appear in π_2. Note that since we have $Q(v_1) = Q(v_2)$, it follows that for every $q \in Q(v_1)$ every element of the set $v_{\pi_2}(q)$ is also an element of view v_1, hence it can form a prefix of π_1. The remaining elements of v_1 can be inserted in π_1 in any order to complete this permutation vector. It follows that for every query q in $Q(v_2)$ (or equivalently in $Q(v_1)$) we have $v_{\pi_2}(q) \subseteq v_{\pi_1}(q)$. Thus $size(v_{\pi_2}(q)) \leq size(v_{\pi_1}(q))$. Also from $v_1 \subset v_2$ we have $size(v_1) \leq size(v_2)$. As a

result, $c_q(v_1, \pi_1) = \frac{size(v_1)}{size(v_{\pi_1}(q))} \leq \frac{size(v_2)}{size(v_{\pi_2}(q))} = c_q(v_2, \pi_2)$ for every query q in $Q(v_2)$ (or equivalently in $Q(v_1)$). $\square$

The above lemma says that if the set of queries that can be answered by view v_1 is the same as the set of queries that can be answered by view v_2, and if view v_1 is a proper subset of view v_2, then for any index π_2 of v_2 we can find an index π_1 of v_1 which is at least as good as π_2 in reducing the cost of answering each query.

Proposition 2. Given an instance of the OLAP-VI problem with input data $(\mathcal{D}, \mathcal{Q}, b)$, if view $v \in V$ has at least one attribute, say attribute a, that is not in any of the queries in $Q(v)$, then the problem has an optimal solution that does not include view v.

Proof. If v is not in the optimal set of views obtained by solving the integer programming model IP1, then clearly removing v from the set V does not affect the optimality of the corresponding solution. So let us assume that view v is in the set of optimal views obtained by solving IP1. Consider view $v' = v \setminus \{a\}$ (the set of all of the attributes in v excluding a). v' is a proper subset of v and since attribute a is not in any of the queries in $Q(v)$, we have $Q(v) = Q(v')$. By Lemma 1, for each index π_v of view v there exists an index $\pi'_{v'}$ of view v' where $c_q(\pi'_{v'}, v') \leq c_q(\pi_v, v)$. Thus substituting v by v' in the optimal set of indexes and replacing each selected index of v, say π_v, by a corresponding index of v' constructed in the manner explained in the proof of Lemma 1, will not increase the total cost of answering the corresponding queries. This implies that an alternative optimal solution can be obtained in which neither view v nor any of its indexes are present. $\square$

We define the reduced set of views, V', as a subset of the set V where each view v in V' is a superset of at least one query in $\mathcal{Q}$, and each attribute of v is an attribute of at least one query in $Q(v)$. Also, any view in V with these two characteristics is in the set V'.

In order to construct the set V', first we remove from the set V every view that is

not a superset of at least one query in $\mathcal{Q}$. Subsequently, from the remaining views we also remove every view whose set of attributes is larger than the union of the set of attributes of the queries that it can answer. The remaining collection of views forms the set V'.

Further, when a view is removed from the search space of views, all of its indexes are also removed automatically from the search space of indexes. As a result, reducing the search space of views directly result in reducing the search space of indexes.

In order to evaluate the computational requirement of determining the set V' we note that there are two methods to construct this set. In the first method we take the union of each combination of r queries (for all $1 \leq r \leq |\mathcal{Q}|$) and add the resulting view to the set V'. In the second method, we consider each of the 2^k views in the view lattice and check whether its set of attributes is equal to the union of the sets of attributes of the queries that it can answer. Note that we always add the raw-data view to the set V'. The computational requirement of the first method is of order $O(2^{|\mathcal{Q}|})$, while the computational requirement of the second method is of order $O(|\mathcal{Q}|2^k)$. Depending on the specific values of k and $|\mathcal{Q}|$, we choose either the first or the second method, whichever results in the smaller computational effort.

3.3 Reducing the Set of Indexes

We now focus on the properties of indexes that appear in an optimal solution for the problem and use these properties to identify a relatively small subset of these indexes for inclusion in our model. In particular, for each view $v \in V'$ we identify a subset $\Pi'(v)$ of $\Pi(v)$ that contains at least one optimal index for this view in the context of any optimal solution for the problem. In order to characterize this restricted collection of indexes $\Pi'(v)$ for each view v, we define and construct a directed graph (digraph) G_v associated with this view.

Each node of the digraph G_v corresponds to a set of attributes that is either equal to the set of attributes of one of the queries in $Q(v)$ or equal to the intersection of

the sets of attributes of two or more queries in $Q(v)$. It follows that associated with each combination of r queries, for $r = 1, 2, \ldots, |Q(v)|$, there is a node in digraph G_v. Two additional nodes are also included in G_v : one node is associated with the view v itself and one node represents the empty set $\emptyset$. For each pair of nodes w_1 and w_2 in G_v, there is an arc from w_1 to w_2 if and only if $w_1 \subset w_2$ and there is no node $w \in G_v$ where $w_1 \subset w \subset w_2$. Note that G_v has a single source $\emptyset$ and a single sink v. For a given view v, the total number of nodes in G_v is at most $min\{2^{|v|}, 2^{|Q(v)|}\}$. In practice, however, the actual number of nodes in G_v may be smaller than this limit. For each view v, the computational requirement for constructing the corresponding digraph G_v is of order $O(min\{8^{|v|}, 8^{|Q(v)|}\})$. In Appendix A, we provide pseudocode for generating the adjacency list for digraph G_v.

Definition 4. Given a view v and its corresponding digraph G_v, let P represent a path that begins at the source node $\emptyset$ and ends at any node w of G_v. For a given index $\pi \in \Pi(v)$ we say that index π is associated with path P if the set of attributes of each and every node in P is a prefix of π.

Definition 5. Given a view v and its corresponding digraph G_v, let P represent a path that begins at the source node $\emptyset$ and ends at a node w_s. Suppose the order of the nodes on path P is $\emptyset$, w_1, w_2, …, w_{i-1}, w_i, w_{i+1}, …, w_s. Given a query $q \in Q(v)$, we say that query q agrees with path P up to node w_i if the set of attributes of each of the nodes $w_1, w_2, \ldots, w_{i-1}$ is a subset of the set of attributes in q, but the set of attributes of the node w_i is not a subset of q.

Lemma 2. Given a view v and its corresponding digraph G_v, if two indexes π_1 and π_2 of view v are associated with the same source-sink path in G_v, then $c_q(v, \pi_1) = c_q(v, \pi_2)$ for every query $q \in Q(v)$.

Proof. Let the source-sink path be denoted by P. Consider the relationship between an arbitrary query $q \in Q(v)$ and the path P. Suppose query q agrees with P up to node z on P. Also, suppose w is the node immediately before z on P. From Definition 5 we have $q \cap w = w$ and $q \cap z = w$. This is because if q has some of the

attributes in $z \setminus w$ then there must exist a node $w' = q \cap z$ on P between nodes w and z. We know that this is not the case, since there is an arc from node z to node w in G_v. On the other hand, since π_1 and π_2 are associated with path P, both of them have all of the attributes in w and all of the attributes in z as their prefix; hence neither of them has any of the attributes in $q \setminus w$ after their first $|w|$ attributes. Thus, the largest subset of q that forms a prefix of π_1 is the same as the largest subset of q that forms a prefix of π_2, i.e., $v_{\pi_1}(q)$=$v_{\pi_2}(q)$. As a result, $c_q(v, \pi_1) = c_q(v, \pi_2)$. □

Lemma 3. Given a view v and its corresponding digraph G_v, if an index π of view v is not associated with any source-sink path in digraph G_v, then there exists another index π' of view v associated with a source-sink path in G_v such that $c_q(v, \pi') \leq c_q(v, \pi)$ for every query $q \in Q(v)$.

Proof. Let $P(\pi)$ be the longest path of G_v that is associated with π. Let w be the last node in $P(\pi)$ and let $r = |w|$. Since π is not associated with any source-sink path of G_v, $P(\pi)$ is not a source-sink path; thus $w \neq v$ and $0 \leq r < |v|$. Suppose the order of attributes in π after the first r attributes is $(a_{r+1}, a_{r+2}, \ldots, a_{|v|})$. We define the set Q_j for $r + 1 \leq j \leq |v|$ as follows: $Q_j = \{q \in Q(v) | w \cup \{a_{r+1}, \ldots, a_j\} \subset q\}$. Let $t = max_{r+1 \leq j \leq |v|}\{j | Q_j \neq \emptyset\}$. We identify node w_j in G_v, for $r + 1 \leq j \leq t$, as the node that corresponds to the intersection of all the queries in Q_j. We have $w \subseteq w_{r+1} \subseteq w_{r+2} \subseteq \ldots \subseteq w_t$. As a result, there exists a source-sink path P that contains all of the nodes on $P(\pi)$ and nodes w_{r+1}, w_{r+2}, …, w_{r+t}. Suppose π' is an index associated with P. Also, suppose the order of the first r attributes in π' is the same as the order of the first r attributes in π. We now show that $c_q(v, \pi') \leq c_q(v, \pi)$ for all $q \in Q(v)$.

Consider any query $q \in Q(v)$. One of the following cases is true:

(1) $(q \cap w) \subset w$

(2) $q = w$

(3) $q \supset w$

If case (1) is true, i.e., q has some (but not all) of the attributes of w, then we have $v_\pi(q) = v_{\pi'}(q)$. This is because both π and π' have all of the attributes of w as their

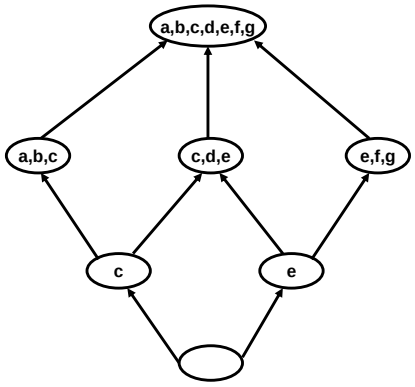


Figure 3.1: Digraph G_v for Example 5.

prefix in the same order. If case (2) is true, then we clearly have $v_\pi(q) = v_{\pi'}(q) = w$. Now suppose case (3) is true. If q has all of the attributes in $\{a_{r+1}, a_{r+2}, \ldots, a_{|v|}\}$, then $q = v$; thus, $v_\pi(q) = v_{\pi'}(q) = v$. On the other hand, if q does not contain a_{r+1}, then $v_\pi(q) = w$. Also, we know that the order of the first r attributes in π and π' is the same. Thus $v_{\pi'}(q) \supseteq w$. As a result, if q does not contain a_{r+1} then we have $v_{\pi'}(q) \supseteq v_\pi(q)$. Now let us consider the case where q contains a_{r+1}, but does not contain all attributes in $\{a_{r+1}, a_{r+2}, \ldots, a_{|v|}\}$. Furthermore, let us assume that q has all of the attributes in $\{a_{r+1}, a_{r+2}, \ldots, a_h\}$, where $r+1 \leq h < t$, but does not include the attribute a_{h+1}. It follows that $q \in Q_h$ and $v_\pi(q) = w \cup \{a_{r+1}, a_{r+2}, \ldots, a_h\}$. Based on the definition of w_h, we have $w \cup \{a_{r+1}, a_{r+2}, \ldots, a_h\} \subseteq w_h$. Thus $v_\pi(q) \subseteq w_h$. From the fact that $q \in Q_h$ we have $w_h \subseteq q$. Since w_h is a node on P and π' is associated with P, all of the attributes of w_h form a prefix of π'. Thus $v_{\pi'}(q) \supseteq w_h$. It follows that for all queries in case (3) we have $v_{\pi'}(q) \supseteq v_\pi(q)$.
We conclude that $v_{\pi'}(q) \supseteq v_\pi(q)$ for any query q in $Q(v)$. Thus we have $size(v_{\pi'}(q)) \geq size(v_\pi(q))$ and consequently $c_q(v, \pi') \leq c_q(v, \pi)$ for every query $q \in Q(v)$. □

From Lemma 2 it follows that if two indexes of view v are associated with the

same source-sink path of digraph G_v, then they have the same effect on reducing the cost of answering each query in $Q(v)$. From Lemma 3 it follows that for each index π of view v that is not associated with any source-sink path of digraph G_v, we can find an index π' of view v which is associated with a source-sink path of G_v and is at least as effective as π in reducing the cost of answering each query in $Q(v)$. We are now ready to define the set $\Pi'(v)$ for each view $v \in V'$.

Definition 6. For a given view v construct the corresponding digraph G_v and determine all distinct source-sink paths in this digraph. For each source-sink path P_i obtained in this manner, determine an associated index π_i. We define $\Pi'(v)$ as the collection of all indexes for view v obtained in this manner. In Appendix B we provide the pseudocode for generating the set $\Pi'(v)$ for a given view v.

Following is a small illustrative example:

Example 5. Consider view $v = \{a, b, c, d, e, f, g\}$ and suppose $Q(v)$ consists of the following queries: $q_1 = \{a, b, c\}$, $q_2 = \{c, d, e\}$, and $q_3 = \{e, f, g\}$. Figure 3.1 represents digraph G_v for v. The source-sink paths in this digraph are as follows:

1. $(\emptyset) \rightarrow (c) \rightarrow (a, b, c) \rightarrow (a, b, c, d, e, f, g)$
2. $(\emptyset) \rightarrow (c) \rightarrow (c, d, e) \rightarrow (a, b, c, d, e, f, g)$
3. $(\emptyset) \rightarrow (e) \rightarrow (c, d, e) \rightarrow (a, b, c, d, e, f, g)$
4. $(\emptyset) \rightarrow (e) \rightarrow (e, f, g) \rightarrow (a, b, c, d, e, f, g)$

An index associated with the first path should have attribute c at its first position, then attributes a and b (in any order), and next attributes d, e, f, and g, in any order after a, b, and c. Thus, the permutation vector (c, a, b, d, e, f, g) is an index associated with the first path. Similarly, we observe that permutation vectors (c, d, e, a, b, f, g), (e, c, d, a, b, f, g), and (e, f, g, a, b, c, d) are indexes associated with the second, the third, and the fourth paths, respectively. Thus, we have:
$\Pi'(v)$=$\{(c, a, b, d, e, f, g), (c, d, e, a, b, f, g), (e, c, d, a, b, f, g), (e, f, g, a, b, c, d)\}$. We note that in this example, $|\Pi'(v)| = 4$ while $|\Pi(v)| = (|v|)!$=5,040. $\square$

From the above discussion it follows that for each view v and for each query $q \in \mathcal{Q}$, the set $\Pi'(v)$ contains at least one index that is at least as effective as any other index in $\Pi(v)$ for answering query q. It follows that we can reduce the search space of indexes in the *OLAP-VI* problem by limiting our search to the smaller collection $\Pi'(v)$ rather than $\Pi(v)$ for each view v, and the resulting (smaller) search space is guaranteed to produce at least one optimal solution for the original problem.

This observation, along with the observations that we made earlier regarding the reduction in the size of the search space of views, could lead to potentially significant reductions in the size of the entire search space of views and indexes for the *OLAP-VI* problem as we shall see in the randomly constructed instances in Chapter 4. Correspondingly we can remove all associated variables and constraints from the integer programming model *IP1*, leading to a smaller model for the problem; we refer to this model as *IP2* as described below.

Modified Integer Programming Model *IP2*.

In this model we use the following notation to represent various restricted subsets of views and indexes and their corresponding collections of subscripts. For each $i = 1, 2, \ldots, m$ let $V_i' = \{v_j \in V' : v_j \supseteq q_i\}$ represent the restricted collection of views that can be used to answer query q_i (where V' is as defined in Section 3.2), and let J_i' represent the corresponding collection of subscripts, i.e., $J_i' = \{j \in J : v_j \in V_i'\}$. Also let J' represent the collection of subscripts of all views in V'. For each view $v_j \in V'$, let $\Pi'(v_j)$ represent the restricted collection of its indexes as defined above, and let L_j' represent the corresponding collection of subscripts, i.e., $L_j' = \{l : \pi_{jl} \in \Pi'(v_j)\}$. We can now write the integer programming model *IP2* using this notation.

$$\min \quad \sum_{i=1}^{m} \sum_{j \in J'_i} \left[d_{ij} s_{ij} + \sum_{l \in L'_j} c_{ijl}.y_{ijl} \right] \qquad \textit{IP2}$$

$$\text{subject to} \quad \sum_{j \in J'_i} \left[s_{ij} + \sum_{l \in L'_j} y_{ijl} \right] = 1 \quad \text{for all } i = 1 \text{ to } m$$

$$\sum_{j \in J'} size(v_j) \left[t_j + \sum_{l \in L'_j} x_{jl} \right] \leq b$$

$$x_{jl} \leq t_j \quad \text{for all } j \in J' \text{ and } l \in L'_j$$

$$s_{ij} \leq t_j \quad \text{for all } i = 1 \text{ to } m \text{ and } j \in J'_i$$

$$y_{ijl} \leq x_{jl} \quad \text{for all } i = 1 \text{ to } m,\ j \in J'_i\ , \text{ and } l \in L'_j$$

$$t_1 = 1$$

$$t_j, s_{ij}, x_{jl}, y_{ijl} \in \{0, 1\} \qquad \forall i, j, l$$

The following theorem follows directly from Propositions 1 and 2, and Lemmas 2 and 3.

Theorem 1. Given an *OLAP-VI* problem with input data $(\mathcal{D}, \mathcal{Q}, b)$, if we define the set V' as in Section 3.2, and the set $\Pi'(v)$ for all $v \in V'$ as in definition 6, then we have the following.
i) every optimal solution of the integer programming model *IP2* is also an optimal solution for the integer programming model *IP1*, and
ii) the integer programming model *IP1* has at least one optimal solution that is also an optimal solution for model *IP2*.

For a given *OLAP-VI* problem, the number of views and indexes considered in model *IP2* can be significantly smaller than the corresponding number in model *IP1*. This is partly due to the fact that the restricted set of views V' can be substantially smaller than the original set V, and partly due to the smaller number of indexes $\Pi'(v)$ for each view v that we consider in model *IP2*. In Chapter 4 we provide numerical

values for a large collection of randomly generated instances of the problem, and observe that the number of distinct paths in G_v, i.e., cardinality of set $\Pi'(v)$, is indeed much smaller than $(|v|)!$ for these instances.

Chapter 4

Experiments with Our Exact Approach

In this chapter we present the results of a computational experiment with the exact approach presented in the previous chapter for solving the OLAP-VI problem. Our objectives in this experiment are:

- evaluating the effectiveness of our approach in reducing the search spaces of views and indexes, and
- evaluating the scalability of the exact model IP2 (see Subsection 3.3) and its effectiveness in solving relatively large instances of the OLAP-VI problem.

To this end, we construct a collection of instances of the OLAP-VI problem of varying sizes using a number of databases of the TPC-H benchmark (see [25]). We then solve each instance using different models and procedures as applicable and report our findings.

We implemented all of our algorithms in C++ and ran them on a PC with a 3GHz Intel P4 processor, 1GB RAM, and a 80GB hard drive running Red Hat Linux

Enterprise 4. We used CPLEX 10 [15] to solve the integer-programming models.

In this chapter first we describe how we construct each of the instances in our experiments. Next we evaluate the effectiveness of our exact approach for solving the OLAP-VI problem. Finally in Section 4.4 we present a summary of our experimental results.

4.1 Constructing Instances

Each instance of the OLAP-VI problem is identified by a given database $\mathcal{D}$, a given collection of queries $\mathcal{Q}$, and a given storage space b. We used two different databases of the TPC-H benchmark (see [25]). More specifically, we used a 7-attribute database and a 13-attribute database to construct the collection of instances in our experiment. We measured the sizes of views in each of these databases using analytical methods.

Aside from the number of attributes in the database, the size of each instance is determined by the number and the makeup of its queries. Within each database we constructed instances of the OLAP-VI problem with the number of queries ranging from 3 to 50. The sizes of the instances that we solved are realistic and comparable to the sizes of the instances used in the related work (cf. [2, 9, 11, 16]).

For each instance we constructed the corresponding collection of queries randomly. More specifically, to construct an instance of the OLAP-VI problem with g queries over a database with k attributes, we first determined the number of attributes in each query as a randomly generated integer (t) between 1 and k-1. Then for each query with t attributes we constructed its actual collection of attributes by randomly generating t distinct integer values between 1 and k. These t integer values uniquely identify the collection of attributes for that query.[1]

The difficulty of solving a specific instance of the OLAP-VI problem depends on the relative value of the storage space b as compared with the size of the raw-data

[1] Consistent with the assumption that we made earlier, we continue to assume that for each query all associated attributes are in its `WHERE` clause and they are compared with constants.

view plus the size of the queries in the set $\mathcal{Q}$. Suppose the value of storage space b is expressed as:

$$b = size(v_1) + \alpha(\sum_{q \in \mathcal{Q}} size(q)) \quad (4.1)$$

where v_1 represents the raw-data view. If $\alpha < 0$ then the problem is infeasible, since the available storage space b is not even sufficient for storing the raw-data view (which is a required selection). If $\alpha = 0$ then the problem is not challenging, since there is only enough space to select the raw-data view. If $\alpha \geq 2$ again the problem is not challenging since the best solution is clearly to materialize the raw-data view plus all queries in the set $\mathcal{Q}$ and an optimal index for each query. Thus, in order for an instance of the view- and index-selection problem to be nontrivial, we need to have $0 < \alpha < 2$.

In our experiments for each instance we set the value of $\alpha = 0.5$, i.e., the storage space limit b is equal to the size of the raw-data view plus one-half of the sum of the sizes of its collection of queries. For some instances we also solved the problem by setting α to several other values between 0 and 2; the pattern of findings did not change very much, although the actual solution did change as expected.

4.2 Reducing the Search Space

In this section we compare the sizes of models *IP1* and *IP2* for some realistic-size instances of the view- and index-selection problem. More specifically, we compare the number of views and indexes in models *IP1* and *IP2* for each instance. Our first collection consists of ten instances over the 7-attribute database and the results are presented in Table 4.1. For each instance (each row) we report the number of queries in that instance and the corresponding number of views and indexes in each of the models *IP1* and *IP2*. Table 4.2 contains similar results for a collection of ten instances over the 13-attribute database. We observe that the number of views and indexes in model *IP2* is significantly smaller than those in model *IP1*, specifically for the relatively smaller instances.

Table 4.1: Comparison of the number of views and indexes in models *IP1* and *IP2* for the instances over the 7-attribute TPC-H database.

instance	number of queries	number of views		number of indexes	
		IP1	*IP2*	*IP1*	*IP2*
1	20	120	45	13,684	913
2	20	118	45	13,681	500
3	20	124	56	13,695	561
4	20	120	47	13,684	521
5	20	120	43	13,684	713
6	20	104	40	13,586	694
7	20	114	42	13,665	418
8	30	126	64	13,698	802
9	40	122	58	13,692	1356
10	50	126	64	13,698	2,117

Table 4.2: Comparison of the number of views and indexes in models *IP1* and *IP2* for the instances over the 13-attribute TPC-H database.

instance	number of queries	number of views		number of indexes	
		IP1	*IP2*	*IP1*	*IP2*
1	10	1,516	27	16,102,349,550	1,218
2	10	6,144	28	16,818,292,374	409
3	10	4,604	25	16,629,888,043	534
4	10	5,376	29	16,801,413,801	358
5	15	5,168	49	16,838,545,683	5,437
6	15	3,280	60	16,718,207,042	7,069
7	20	6,608	165	16,921,115,940	15,460
8	30	7,456	305	16,925,556,971	3,061,155,903
9	40	7,022	556	16,925,054,570	3,177,400,892
10	50	7,092	725	16,924,507,774	3,166,128,428

Table 4.3: The time required to solve the instances over the 7-attribute database using model IP2, and the time required to build the search space of views and indexes.

instance	number of queries	time to build search space (sec.)	IP2 execution time (sec.)
1	20	1.45	10.14
2	20	1.35	2.17
3	20	1.29	3.66
4	20	1.27	1.93
5	20	1.52	3.26
6	20	1.40	2.99
7	20	1.31	4.01
8	30	1.52	13.82
9	40	1.98	50.57
10	50	1.91	500.50

4.3 Scalability of the Exact Model IP2

In this section we evaluate the scalability of model IP2 to solve instances of the OLAP-VI problem. To do so, we solve each of the instances of the two sets of experiments explained in the previous section using model IP2, and report the total time required to solve each of these instances in Tables 4.3 and 4.4. We do not report similar results for model IP1, since the corresponding execution times are excessive and consistently larger than those for IP2.

As observed in these two tables, we could solve all instances over the 7-attribute database using model IP2. However, we could not solve those large instances over the 13-attribute database where the number of queries is 20 or more as the computer ran out of memory. We also solved many other instances over these two databases. We could always solve the instances over the 7-attribute database within one hour. However, we could never solve instances with more than 30 queries over the 13-attribute database.

In the instances that we could solve model IP2, we also separately measured the time required to build the search space of views and indexes of model IP2 and the time required to solve this model, and report them in Tables 4.3 and 4.4. We observed that

Table 4.4: The time required to solve the instances over the 13-attribute database using model IP2, and the time required to build the search space of views and indexes. We could not solve the last four instances as the computer ran out of memory.

instance	number of queries	time to build search space (sec.)	IP2 execution time (sec.)
1	10	1.71	2.10
2	10	1.32	1.96
3	10	1.4	1.96
4	10	1.31	1.83
5	15	15.81	287.34
6	15	18.92	520.74
7	20	151.31	memory shortage
8	30	511.82	memory shortage
9	40	844.21	memory shortage
10	50	1472.13	memory shortage

for all instances the time required to build the search space of model IP2 is smaller than the time required to solve the corresponding model IP2, and for the larger instances with larger execution times this difference is quite significant (instances 8, 9, and 10 in Table 4.3 and instances 5 and 6 in Table 4.4).

4.4 Summary of Observations

From the above experiments we observe that the search space of views and indexes in model IP2 is significantly smaller than the search space of views and indexes in model IP1. This allows us to solve realistic-size instances of the OLAP-VI problem optimally using model IP2 while we cannot solve these instances using model IP1. Also, we observe that there are many realistic cases that due to their large sizes we are not able to solve the problem within a reasonable execution time, even when we use the smaller model IP2. In order to be able to solve such larger instances, we developed several heuristic approaches that we present in the next chapter.

Chapter 5

Inexact Methods

In this chapter we propose three strategies to reduce the size of the integer programming model for the view- and index-selection problem. The primary objective of these strategies is to reduce the size of the IP model for larger instances of the OLAP-VI problem so that it can be solved within reasonable execution time. The downside of this approach, however, is that we can no longer guarantee that an optimal solution of the resulting integer programming models is an optimal solution of the original OVIP-VI problem.

In the first strategy, we limit the number of indexes for each view $v \in V'$ to a collection of promising alternatives which is a subset of $\Pi'(v)$. This collection is significantly smaller than the corresponding collection in model IP2; hence the corresponding integer programming model is also smaller. In the second strategy, again we limit the choice of indexes for each view $v \in V'$ to a collection of promising alternatives which is also significantly smaller than the corresponding collection in model IP2. However, unlike the first strategy, this collection is not necessarily a subset of $\Pi'(v)$. In the third strategy, we limit the choice of views to a collection of promising alternatives. This collection of views is significantly smaller than the corresponding collection in model IP2, hence again the corresponding integer programming model

is smaller (than IP2). The trade off among these three strategies is the size of the instances that we can solve verses the quality of the solutions that we obtain. As we will demonstrate in Chapter 6 through our experimental results, we can get solutions with better qualities using the first strategy while we can solve larger instances using the second and the third strategies. All of these strategies would allow us to solve the instances of the view- and index-selection problem that are much larger than those instances that we are able to solve with our exact methods.

We propose three specific methods for carrying out these strategies, resulting in three integer programming models that we refer to as IPN, IPNIRp, and IPVs. In model IPN, the set of indexes for each view $v \in V'$ is a subset of the corresponding set $\Pi'(v)$. Model IPNIRp is an extension of model IP2 with significantly smaller number of indexes for each view $v \in V'$. In model IPVs, the set of views is limited to a subset of V' which we refer to it as V''. Also, in model IPVs, the set of indexes for each view $v \in V''$ is a proper subset of the corresponding set $\Pi'(v)$.

5.1 Model IPN

In the approach we propose in this section, for a given instance of the OLAP-VI problem with input data $(\mathcal{D}, \mathcal{Q}, b)$, we limit the number of indexes for each view $v \in V'$ to a relatively small positive integer that we refer to as N_v. We select $N_v = |Q(v)|$. Other values of N_v can also be used in this context. Our choice of the value of N_v, i.e., $|Q(v)|$, is inspired by the fact that it is an upper bound on the number of indexes for view v at the optimal solution of model IP2 presented in Section 3.3 (note that each query $q \in Q(v)$ can be answered optimally with respect to v by at most one index of v).

For each view $v \in V'$, we determine a set $\Pi''(v)$ which is a collection of N_v promising indexes among all indexes over view v, and remove all other indexes over view v from the integer programming model. The resulting integer programming model is smaller than model IP2, although it has the same structure, i.e., same

type of variables and constraints. We refer to this model as IPN and define it more specifically later in this section.

We select the N_v indexes associated with each view $v \in V'$ in a greedy manner as follows:

$\pi_v^1 = argmin_{\pi \in \Pi'(v)} \sum_{q \in Q(v)} c_q(v, \pi)$

$\pi_v^2 = argmin_{\pi \in \Pi'(v)} \sum_{q \in Q(v)} \min\{c_q(v, \pi), c_q(v, \pi_v^1)\}$

$\pi_v^3 = argmin_{\pi \in \Pi'(v)} \sum_{q \in Q(v)} \min\{c_q(v, \pi), c_q(v, \pi_v^1), c_q(v, \pi_v^2)\}$

...

$\pi_v^{N_v} = argmin_{\pi \in \Pi'(v)} \sum_{q \in Q(v)} \min\{c_q(v, \pi), c_q(v, \pi_v^1), c_q(v, \pi_v^2), \ldots, c_q(v, \pi_v^{N_v - 1})\}$

and define the set $\Pi''(v) = \{\pi_v^1, \pi_v^2, \ldots, \pi_v^{N_v}\}$.

Model IPN

We define model IPN similarly to model IP2, except that we use the set of indexes $\Pi''(v)$ in place of $\Pi'(v)$ for each view $v \in V'$. Note that while the number of variables and constraints in model IPN can be significantly smaller than the corresponding numbers in model IP2, the optimal solution of this model is no longer guaranteed to be optimal for the original problem.

In the remainder of this section, we present an efficient algorithm which we refer to as Algorithm IPNIDX to determine the set $\Pi''(v)$ for each view $v \in V'$.

Algorithm IPNIDX

The pseudocode for Algorithm IPNIDX is displayed in the next page. This algorithm is iterative: In each iteration we find one index. That is, in the first iteration we find π_v^1, in the second iteration we find π_v^2, and so on. In each iteration, we consider the nodes of digraph G_v in topological order. For each node w we find an order for the attributes of w, based on the order of the attributes of one of its parent nodes. We refer to this order as $perm(w)$. The last node considered in this topological order is the sink node v, and we declare the corresponding order $perm(v)$ as the index

selected in that iteration.

At the end of each iteration, we update the value of $MCS(q)$ for each $q \in Q(v)$, where $MCS(q)$ is the minimum cost of answering query q using the indexes selected so far. At the beginning of the first iteration, $MCS(q) = size(v)$ for all queries in $Q(v)$.

To find the order of attributes of node w in each iteration, first we consider the set of queries that affect the order of the attributes of w, i.e., queries in the set $Q' = \{q \in Q_{temp} | q \cap w \neq \emptyset \text{ and } q \cap w \neq w\}$. Note that query $q \in Q(v)$ is in Q_{temp} if its set of attributes does not form a prefix of any index selected so far in the algorithm. Also, from the property of indexes in $\Pi'(v)$ we have $perm(w) = (perm(u), arb(w \setminus u))$, where u is one of the parent nodes of node w in G_v, and $arb(w \setminus u)$ is an arbitrary order of the attributes in $w \setminus u$. Thus, we need to find the parent of w that yields the minimum total cost of answering the queries in Q'. Since we consider the nodes in the topological order, at the time of computing $perm(w)$ we know the value of $cost(u, q)$ for each parent node u and for each query in Q'; thus we can compute $cost(u) = \sum_{q \in Q'} min\{cost(u, q), MCS(q)\}$ for each parent node u of node w.

Given the digraph G_v for view v, the computational requirement of Algorithm IPNIDX is $O(N_v \times |Q(v)| \times 4^{min\{|Q(v)|,|v|\}})$. Too see this, note that the *while* loop of Algorithm IPNIDX is repeated N_v times. The *for* loop in the while loop is repeated at most $(the\ number\ of\ nodes)^2 \times |Q(v)|$ times. As stated earlier, the maximum number of nodes in G_v is $2^{min\{|Q(v)|,|v|\}}$. Thus, the computational requirement of Algorithm IPNIDX is of the order of $O(|Q(v)|^2 \times 4^{min\{|Q(v)|,|v|\}})$.

Note that the number of variables and constraints in model IPN can be significantly smaller than the corresponding number in model IP2, but the optimal solution of this model is no longer guaranteed to be optimal for the original problem. In the next chapter we present the results of a computational experiment in which we evaluate the execution time and the quality of solutions obtained using this approach for a relatively large collection of instances of OLAP-VI problem. In Appendix C we provide an example for constructing the set $\Pi''(v)$ for a given view v.

Algorithm 1: IPNIDX

An efficient algorithm to find the elements of the set $\Pi''(v)$

Input : view v, digraph G_v, and the set of queries $Q(v)$
(See Section 3.3 for construction of G_v.)

Output: all elements of the set $\Pi''(v)$

begin

$MCS(q) = size(v)$ for all $q \in Q(v)$
$cost(\emptyset, q) = size(v)$ for all $q \in Q(v)$
$Q_{temp} \leftarrow Q(v)$
$r \leftarrow 1$
while $r \leq N_v$ **do**
  for *each node* w *(other than* $\emptyset$*) in* G_v *in topological order* **do**
    set $Q' \leftarrow \{q \in Q_{temp} | q \cap w \neq \emptyset$ and $q \cap w \neq w\}$
    for *each node* u *which is a parent node of* w **do**
      $cost(u) = \sum_{q \in Q'} \min\{cost(u, q), MCS(q)\}$
    $\hat{u} = argmin_{u \in parent(w)}\ cost(u)$
    $perm(w) = perm(\hat{u})$,arb$(w \setminus \hat{u})$
    $cost(w, q) = c_q(v, perm(w))$ for each $q \in Q_{temp}$
  return $perm(v)$
  $MCS(q) = \min\{cost(v, q), MCS(q)\}$ for each $q \in Q_{temp}$
  $Q_{temp} \leftarrow \{q \in Q_{temp} |\ v_{perm(v)}(q) \neq q\}$.
  $r \leftarrow r + 1$

end

The notation $arb(w)$ in this algorithm represents an arbitrary order of attributes of set w, the set $parent(w)$ represents the set of parent nodes of node w, and $perm(w)$ represents the order of the attributes of node w.

5.2 Model IPNIRp

Through studying the structure of the OLAP-VI problem, we observed that in most of the instances the order of the first few attributes of each index is much more important than the order of the rest of the attributes of that index. When the first few attributes of index π are common to the largest number of queries in $Q(v)$, the index π tends to be more effective in reducing the total cost of answering the queries. Therefore, another promising approach to reduce the size of the integer programing model is to remove from the digraph G_v every node that is an intersection of only a small number of queries in $Q(v)$. This way, the number of nodes and consequently the number of source-sink paths and the number of indexes in the search space of indexes of each view v would decrease (compared to $\Pi'(v)$). Based on these observations, in this section we introduce another reduced IP model which we refer to as model IPNIRp. To do so, first we define digraph G_v^p for each view $v \in V'$ and the reduced set of indexes $\Pi'_p(v)$.

For each view v we define digraph G_v^p as follows. Each node of digraph G_v^p corresponds to a set of attributes that is the intersection of the sets of attributes of p or more queries in $Q(v)$ (note that associated with each combination of r queries, for $r = p, p+1, \ldots, |Q(v)|$, there is a node in digraph G_v^p.) Two additional nodes associated with v and $\emptyset$ are also included in the set of nodes for G_v^p. Similar to G_v, for each two nodes w_i and w_j in G_v^p, there is an arc from w_i to w_j if and only if $w_i \subset w_j$ and there is no node $w_k \in G_v^p$ where $w_i \subset w_k \subset w_j$. Note that G_v^p has a single source $\emptyset$ and a single sink v.

Definition 7. (*Reduced Set of Indexes* $\Pi'_p(v)$) For each view v construct the corresponding digraph G_v^p and determine all distinct source-sink paths in this digraph. For each source-sink path P_i obtained in this manner, determine an associated index π_i. We define $\Pi'_p(v)$ as the collection of indexes for view v obtained in this manner.

Model IPNIRp

We define model IPNIRp similarly to model IP2, except that we use the set of indexes $\Pi'_p(v)$ in place of $\Pi'(v)$ for each view $v \in V'$.

Note that p is the parameter of model IPNIRp. As we increase the value of p, most likely the number of nodes in G_v^p would decrease; thus the number of source-sink paths in G_v^p would decrease as well. As a result, as we increase the value of p, most likely the number of indexes in $\Pi'_p(v)$ would decrease, and IPNIRp would become smaller. Thus when the value of parameter p is large enough, we can solve very large scale instances using model IPNIRp. However, note that if $p \geq 2$, the optimal solution of model IPNIRp is not necessarily optimal for the original OLAP-VI problem. (Note that when $p = 1$ then IPNIRp is the same as IP2 and provides optimal solutions.)

A reasonable strategy to solve an instance of the OLAP-VI problem using model IPNIRp is to decrease the value of parameter p as much as time allows.

5.3 Model IPVs

In our experiments (as we will present in Section 6.3) we observed that although we can solve significantly larger instances using model IPN compared to the instances that we can solve using model IP2, there are still large instances that we could not solve using model IPN within our time limit of one hour. In this section we propose a strategy to reduce the size of the integer programming model for the view- and index-selection problem further. In this strategy, we limit the choice of views to a collection of promising alternatives. Depending on the value of parameter s, this collection can be significantly smaller than the corresponding collection in our previous models IP2, IPN, and IPNIRp; hence the corresponding integer programming model can be smaller.

The reduced set of views

We define the set V_s'' as the reduced set of views where each view in V_s'' is the union of at most s queries in $\mathcal{Q}$. It follows that with respect to each r queries in $\mathcal{Q}$, $1 \leq r \leq s$, there is a view in V_s'' which is the union of these r queries.

Note that as we increase the value of parameter s, most likely the number of views in V_s'' would increase as well, and when $s = |\mathcal{Q}|$ we have $V_s'' = V'$.

Model IPVs

We define the integer programming model IPVs similarly to model IPN, except that we use the reduced set of views V_s'' in place of V'. Also, as in model IPN, in model IPVs we consider the reduced set of indexes $\Pi''(v)$ to be the search space of indexes for each view $v \in V_s''$.

Although when $s = |\mathcal{Q}|$ we have $V_s'' = V'$, our experimental results in Section 6.3 show that when s is smaller that $|\mathcal{Q}|$, $|V_s''|$ is significantly smaller that $|V'|$. As a result when the value of parameter s is small enough, we can solve very large-scale instances using model IPVs. A reasonable strategy to solve an instance of the OLAP-VI problem using model IPVs is to increase the value of parameter s as large as time allows.

The computational requirement of obtaining V' is of order of $O(\sum_{i=1}^{s} C \binom{|Q(v)|}{i})$.

Chapter 6

Experiments with Our Inexact Methods

In this chapter we present the results of computational experiments with the approaches presented in the previous chapter for solving the OLAP-VI problem. Our objectives in these experiments are (1) to evaluate the scalability of the inexact models IPN, IPNIRp, and IPVs (see Chapter 5) and the quality of the solutions obtained by these models, and (2) to compare the effectiveness of the proposed model IPN with other heuristic procedures in the open literature.

To this end, we construct a collection of instances of the OLAP-VI problem in the same way that we explained in Section 4.1. We then solve each instance using different models and procedures as applicable and report our findings. In this chapter, not only we used the 7-attribute and the 13-attribute TPC-H databases (described in Section 4.1), but also we used a larger size TPC-H database with 17 attributes.

As stated in Chapter 4, we implemented all of our algorithms in C++ and ran them on a PC with a 3GHz Intel P4 processor, 1GB RAM, and a 80GB hard drive running Red Hat Linux Enterprise 4. For comparative purposes, we also implemented

a modified version of the approach for selecting views and indexes proposed in [2], which we refer to as *algorithm ACN*, and experimentally compared the results of our inexact approach IPN with the results obtained from this algorithm. We also implemented a heuristic approach for selecting views and indexes proposed in [13] which we refer to as *algorithm GHRU*. We experimentally evaluated this algorithm and compare it with our approaches too. We implemented algorithms ACN and GHRU in C++ and ran on the same platform as the rest of our algorithms.

The rest of this chapter is organized as follows: In Sections 6.1 through 6.3 we present a detailed account of our experiments and analysis on our inexact approaches explained in Chapter 5. In Section 6.4 we experimentally compare our inexact approach IPN with algorithm ACN for selecting views and indexes. Finally in Section 6.5 we evaluate algorithm GHRU and compare it with our proposed approaches.

6.1 Experiments with the Inexact Model IPN

We pursue the following goals in this section:

- experimentally evaluating the quality of the solutions obtained using model IPN.
- study the effect of the storage space bound on the quality of the solutions obtained from model IPN.
- evaluating the performance of model IPN when queries in the workload are from certain levels of the view lattice.

As mentioned earlier, we set the value of $N_v = |Q(v)|$ in algorithm IPNIDX. In fact, we solved a collection of instances for several values of N_v such as 1, $0.1|Q(v)|$, $0.2|Q(v)|$, $0.3|Q(v)|$, $0.5|Q(v)|$, and $1.5|Q(v)|$. In our analysis we observed that the choice of value $|Q(v)|$ for parameter N_v in algorithm IPNIDX would consistently result in solutions of acceptable qualities for all instances in this collection.

Table 6.1: The value of cost obtained using model IPN and the optimal value of cost (obtained from solving model IP2) for instances over the 13-attribute database. α is defined in the context of Formula 4.1 for storage space.

instance number	number of queries	α=0.1		α=0.3	
		IPN	optimal	IPN	optimal
1	7	1798880.00	1798880.00	12041.20	12041.20
2	9	2572900.00	2572900.00	71.98	71.97
3	11	2398520.00	2398520.00	299942.00	299942.00
4	13	2698330.00	2698330.00	300028.00	299956.00
5	15	3297958.00	3297958.00	299892.00	299857.00

instance number	α=0.5		α=1	
	IPN	optimal	IPN	optimal
1	49.62	49.62	7.00	7.00
2	9.24	9.24	9.00	9.00
3	11.26	11.13	11.00	11.00
4	13.08	13.00	13.00	13.00
5	15.02	15.01	15.00	15.00

In the first set of experiments, we evaluate the quality of the solutions obtained from model IPN for different values of storage space bound. We solve five instances over the 13-attribute database. These instances have 7, 9, 11, 13, and 15 queries. We solve each instance with several values of the storage space limit as depicted by the value of α: 0.1, 0.3, 0.5, and 1. (α is defined in the context of Formula 4.1 in Section 4.1.)

In Table 6.1 for each instance and each value of α we present the value of cost obtained using model IPN as well as the optimal value of cost (obtained from model IP2). In these instances we observe that regardless of the value of α, IPN provides solutions which are either optimal or very close to optimal.

In the next set of experiments we evaluate the quality of solutions obtained using model IPN for the cases where all of the queries in the workload are from certain levels of the view lattice. We solve twelve instances over the 13-attribute database. In the first two instances the number of attributes in each query is a random number

Table 6.2: The value of cost obtained by model IPN and the optimal value of cost (obtained from solving model IP2) for instances on the 13-attribute database. Queries in instances 1 and 2 are from the upper levels of the view lattice, queries in instances 3-7 are from the middle levels of the view lattice, and queries in instances 8-12 are from the lower levels of the view lattice.

instance number	number of queries	α=0.1		α=0.3	
		IPN	optimal	IPN	optimal
1	7	2098700.00	2098700.00	7.10	7.06
2	9	2698330.00	2698330.00	11.89	11.47
3	7	2098700.00	2098700.00	35982.40	35982.40
4	9	2698330.00	2698330.00	611639.00	611639.00
5	11	300060.00	300060.00	11.29	11.15
6	13	899923.00	899923.00	188.84	188.79
7	15	309819.00	309818.00	626.55	625.22
8	7	599633.00	599633.00	599633.00	599633.00
9	9	1705540.00	1705540.00	602696.00	602696.00
10	11	1533840.00	1533830.00	1249140.00	1249140.00
11	13	1199270.00	1199270.00	960973.00	960973.00
12	15	2452170.00	2452170.00	600112.00	600112.00

instance number	number of queries	α=0.5		α=1	
		IPN	optimal	IPN	optimal
1	7	7.00	7.00	7.00	7.00
2	9	9.00	9.00	9.00	9.00
3	7	7.80	7.79	7.00	7.00
4	9	10.36	9.75	9.00	9.00
5	11	11.02	11.00	11.00	11.00
6	13	13.07	13.01	13.00	13.00
7	15	15.59	15.05	15.01	15.00
8	7	539242.00	539242.00	299820.00	299820.00
9	9	197.62	197.62	10.10	10.10
10	11	331397.00	331397.00	49.00	49.00
11	13	299948.00	299945.00	13.05	13.03
12	15	77.11	77.11	15.08	15.08

between 9 and 12, so queries are from the upper levels of the view lattice. In the next five instances the number of attributes in each query is a random number between 5 and 8, so queries are from the middle levels of the view lattice. Finally, in the last five instances the number of attributes in each query is a random number between 1 and 4, so queries are from the lower levels of the view lattice. We solve each instance for different values of α: $\alpha = 0.1$, 0.3, 0.5, and 1. We also solve each instance optimally (using model IP2). In Table 6.2, for each instance we present the cost obtained by model IPN and the optimal value of cost obtained from solving model IP2. In this table we observe that regardless of the position (level) of the queries in the view lattice, model IPN provides solutions which are close to optimal. (When queries are from the upper levels of the view lattice, in our case when each query has between 9 and 12 attributes, for the instances where the number of queries is more than nine we could not obtain the optimal solution using model IP2 due to the memory shortage, hence we do not include such instances in Table 6.2. Note that when queries are from the upper levels of the view lattice, they have many attributes in common which results in large digraph G_v's for many views, hence a larger IP model.)

We perform two other sets of experiments to evaluate the performance of model IPN for larger instances. These sets of experiments consist of solving a collection of ten instances over the 7-attribute database and a collection of ten instances over the 13-attribute database with the number of queries ranging between 10 and 50. In these instances the value of α is 0.5, and queries are selected randomly (these instances are exactly the same as instances that we described in Chapter 4). We report our findings in Tables 6.3 and 6.4, respectively. For each instance we report (1) the number of queries, (2) the optimal value of the corresponding integer programming models IP2 and IPN, (3) the total time required to solve each instance using each of models IP2 and IPN, and (4) the number of indexes in model IP2 and model IPN. Note that for each instance, the number of views in models IP2 and IPN is the same and it is reported in Tables 6.3 and 6.4. For these collections of instances, we make the following observations:

Table 6.3: Comparison of (1) the value of cost obtained from solving model IPN with the value of cost obtained from solving model IP2 (optimal), (2) the total time required to solve each of the instances using model IPN with the total time required to solve the same instance using model IP2, and (3) the number of indexes in the search space of indexes of model IPN with the corresponding number in model IP2 for each of the instances over the 7-attribute database.

instance	number of queries	IP2 cost (optimal)	IPN cost	IP2 time (sec.)	IPN time (sec.)
1	20	20.25	20.25	10.14	0.73
2	20	20.41	22.17	2.17	0.62
3	20	20.29	20.54	3.66	0.75
4	20	20.42	20.66	1.93	0.76
5	20	20.41	20.52	3.26	0.74
6	20	20.17	21.00	2.99	0.66
7	20	21.05	21.16	4.01	1.59
8	30	30.29	30.30	13.82	1.69
9	40	40.35	40.35	50.57	1.16
10	50	50.16	50.30	500.5	3.56

instance	number of queries	number of views in V'	number of indexes in IP2	number of indexes in IPN
1	20	45	913	150
2	20	45	500	132
3	20	56	561	172
4	20	47	521	159
5	20	43	713	161
6	20	40	694	113
7	20	42	418	125
8	30	64	802	259
9	40	58	1356	324
10	50	64	2117	408

Table 6.4: Comparison of (1) the value of cost obtained from solving model IPN with the value of cost obtained from solving model IP2 (optimal), (2) the total time required to solve each of the instances using model IPN with the total time required to solve the same instance using model IP2, and (3) the number of indexes in the search space of indexes of model IPN with the corresponding number in model IP2 for each of the instances over the 13-attribute database.

instance	number of queries	IP2 cost (optimal)	IPN cost	IP2 time (sec.)	IPN time (sec.)
1	10	10.00	10.00	2.10	0.65
2	10	10.07	10.07	1.96	0.61
3	10	10.00	10.00	1.96	0.61
4	10	10.29	10.64	1.83	0.62
5	15	15.11	15.44	287.34	0.93
6	15	15.12	15.23	520.74	0.82
7	20	-	20.63	memory shortage	1.49
8	30	-	30.11	memory shortage	10.17
9	40	-	40.14	memory shortage	21.71
10	50	-	50.01	memory shortage	291.57

instance	number of queries	number of views in V'	number of indexes in IP2	number of indexes in IPN
1	10	27	1218	77
2	10	28	409	77
3	10	25	534	72
4	10	29	358	88
5	15	49	5,437	159
6	15	60	7,069	203
7	20	165	15,460	619
8	30	305	3,061,155,903	1207
9	40	556	3,177,400,892	2464
10	50	725	3,166,128,428	4345

- The time required to solve each of the instances using the inexact model IPN is significantly smaller than the corresponding time for model IP2 (especially for the larger instances). Furthermore, the quality of the solution obtained via model IPN is close to optimal for those instances where we know the optimal solution (obtained via the exact model IP2). For this collection of instances, on average the value of cost obtained from model IPN is 1% more than the corresponding optimal value obtained from model IP2. The maximum deviation of the value of cost obtained by solving model IPN from the optimal value is 9% (instance 2 in Table 6.3). For the remaining instances where we do not have the optimal value (instances 7-10 in Table 6.4), the cost obtained via IPN is at most 4% larger than the corresponding lower bound (i.e., the number of queries).[1]

- In all instances we observe that the number of indexes in model IPN is significantly smaller than the corresponding number in model IP2, especially for larger instances.

As in the experiments with model IP2, in these instances we also measured the time required to build the search space of views and indexes for model IPN using algorithm IPNIDX and the time required to solve model IPN, separately. We observed that for the larger instances where the execution time is significant, the time required to build the search space of model IPN is significantly smaller than the time required to solve the corresponding model IPN.

We also solved a larger instance on the 13-attribute database with 100 queries using model IPN. The total time required to solve this instance using model IPN was 2,674 seconds, and the value of cost obtained from this model is 100.06. We could not solve model IP2 for this instance as the computer ran out of memory.

[1]Consistent with the assumption that we made earlier, we continue to assume that for each query all associated attributes are in its `WHERE` clause and they are compared with constants. As $v_\pi(q) \subseteq v$, we have $size(v_\pi(q)) \leq size(v)$; thus $c_q(v,\pi) = \frac{size(v)}{size(v_\pi(q))} \geq 1$. It follows that the cost of answering each query would be at least 1. As a result, for every instance in our experiments, the number of queries forms a lower bound for its associated cost.

Table 6.5: Comparison of the value of cost obtained from model IPNIRp with the value of cost obtained from model IPN for different values of p for a large instance on the 13-attribute database with 100 queries.

p	IPNIRp cost	IPN cost	IPNIRp time (sec.)	IPN time (sec.)	number of indexes in IPNIRp	number of indexes in IPN
20	100.12	100.06	356.30	2674	4043	22100
25	100.44		64.79		2936	
30	103.20		40.99		2743	
35	156.00		48.52		2667	
40	156.40		34.95		2655	
45	156.00		33.02		2651	
50	364.43		36.90		2639	

6.2 Experiments with the Inexact Model IPNIRp

In this section we evaluate the quality of the solutions obtained from model IPNIRp described in Section 5.2. Also, we study the impact of parameter p in this model on the quality of the solutions.

Consider the large instance (with 100 queries) that we mentioned at the end of the previous section. We solved this instance using model IPNIRp for p=20, 25, 30, 35, 40, 45, and 50. The results are presented in Table 6.5. As we can see in this table, when p=20 and 25, the cost obtained from model IPNIRp is close to the cost obtained from model IPN. Also we observe that for all values of p, the time required for solving this large instance using model IPNIRp is significantly smaller than the time required for solving this instance using model IPN (as expected). Furthermore, we observe that mostly, as we increase the value of parameter p, the number of indexes in the search space of model IPNIRp and the time required for solving IPNIRp decrease, but the value of cost increases.

Our goal in the next set of experiments is to find those values of parameter p which result in solutions with acceptable qualities. To this end, we construct and solve a collection of eighteen relatively large instances (with number of queries between 50

Table 6.6: The value of $p_{0.01}$ for instances on the 13-attribute database.

instance number	number of queries	$p_{0.01}$	IPNIR$p_{0.01}$ time (sec.)	IPN time (sec.)	$\frac{IPNtime}{IPNIRp_{0.01}time}$
1	50	15	10.55	101.53	9.6
2	50	15	16.09	108.64	6.8
3	50	15	8.42	66.62	7.9
4	60	15	10.32	89.99	8.7
5	60	10	47.74	83.92	1.8
6	60	20	5.35	61.88	11.6
7	70	10	66.48	384.35	5.8
8	70	20	7.75	304.88	39.3
9	70	15	7.12	158.98	22.3
10	80	20	12.74	277.67	21.8
11	80	20	52.39	423.09	8.1
12	80	15	14.27	308.7	21.6
13	90	25	14.10	282.77	20.1
14	90	20	23.18	449.66	19.4
15	90	20	17.06	505.76	29.6
16	100	20	43.06	287.08	6.7
17	100	20	20.99	485.64	23.1
18	100	20	28.52	667.93	23.4

and 100) over the 13-attribute TPC-H database. For each instance, we decrease the value of p in IPNIRp from 30 to 25 to 20, ... until v(IPNIRp)$\leq$ 1.01v(IPN), where v(IPNIRp) represents the value of cost obtained from model IPNIRp and v(IPN) represents the value of cost obtained from model IPN. We use the notation $p_{0.01}$ to represent the largest value of parameter p among values $5, 10, 15, 20, 25, 30$ that results in v(IPNIRp)$\leq$ 1.01v(IPN). In Table 6.6 for each instance we report $p_{0.01}$, the time required to solve that instance using model IPN, and the time required to solve that instance using model IPNIR$p_{0.01}$. From this table, we observe that in this experiment, $10\leq p_{0.01} \leq 25$, and on average the time required to solve each instance using model IPNIR$p_{0.01}$ is 16 times less than the time required to solve that instance using the corresponding model IPN.

We also solved three larger instances with 50 queries over the 17-attribute database

Table 6.7: The value of $p_{0.01}$ for instances on the 17-attribute database.

instance number	number of queries	$p_{0.01}$	IPNIRp with $p = p_{0.01}$ time (sec.)	IPN time (sec.)	$\frac{IPNtime}{IPNIRp_{0.01}time}$
1	50	15	18.95	514.74	27.2
2	50	10	184.05	1723.09	9.4
3	50	20	23.34	927.14	39.7

using IPN and IPNIRp with different values of p. For these instances we report $p_{0.01}$, the time required to solve each instance using model IPN, and the time required to solve each instance using model IPNIRp with $p = p_{0.01}$ in Table 6.7. From this table, we observe that for these instances, $15 \leq p_{0.01} \leq 20$, and the time required to solve each instance using model IPNIRp with $p = p_{0.01}$ is between about 10 to 40 times less than the time required to solve the corresponding model IPN. We could not solve instances with 60 queries and more over this database (the 17-attribute database) using IPN in our time limit of one hour.

In our above experiments we observe that we can solve instances using model IPNIRp with $p = p_{0.01}$ significantly faster than when we use model IPN (and of course model IPNIRp with $p = p_{0.01}$ provides solutions of acceptable qualities with respect to the solutions obtained from model IPN) . As a result, we can solve many large instances using model IPNIRp that we are not able to solve in our time limit using model IPN. An example of such instances is presented in Table 6.8 which is an instance with 70 queries on the 17-attribute database. As we observe in this table, for this instance the search space of indexes in model IPN is relatively large and we could not solve this instance using model IPN. However, we could solve this instance using model IPNIRp when p=20 in 660.96 seconds.[2] (The value of cost obtained from IPNIRp is within 1% of lower bound for the value of cost.)

[2]We need to mention that we could not solve this instance using model IPNIRp when p=10 in one hour.

Table 6.8: Solving a large instance with 70 queries on the 17-attribute database using model IPNIRp with p=20. We could not solve this instance using p=10 in one hour.

IPNIRp(p=20) cost	IPNIRp(p=20) time (sec.)	number of indexes in IPNIRp(p=20)	number of indexes in IPN
70.560	660.96	11,485	88,649

6.3 Experiments with the Inexact Model IPVs

In this section we evaluate the quality of the solutions obtained from model IPVs. Also, we study the effect of parameter s on the quality of the solutions obtained using this model. To this end, we solve four instances on the largest database, i.e., the 17-attribute database using model IPVs with different values of parameter s, and present the results in Tables 6.9 through 6.12 (each table corresponds to a separate instance). In each of these instances we have 50 randomly generated queries. In Tables 6.9, 6.10, 6.11, and 6.12 for each value of parameter s, we present the value of cost obtained from model IPVs, the number of views and indexes in the search space of model IPVs, and the time required for solving each instance using model IPVs. Note that in these instances when $s = 50$ model IPVs is the same as model IPN. As we can observe in Tables 6.10 and 6.12 we cannot solve the second and the fourth instances using model IPN in one hour.

From Tables 6.9 through 6.12 we observe that for those instances that we were able to solve model IPN (instances 1 and 3), when $s = 1$ model IPVs provides solutions which are close to the solution of model IPN. Furthermore, we observe that the value of cost obtained via model IPVs (as a function of s) is monotonically non-increasing as expected.

To further evaluate the quality of the solutions obtained from model IPVs when $s = 1$, we solved the 18 instances of Table 6.6 using model IPVs for $s = 1$. In Table 6.13 for each instance we present the value of cost and the time required to solve that instance using model IPVs (s=1), as well as the value of cost and the time required to solve that instance using model IPN. From this table we observe that in

Table 6.9: The result of solving instance 1 over the 17-attribute database using model IPVs for different values of s.

s	Number of views in $\lvert V_s'' \rvert$	Number of indexes in IPVs	IPVs cost	IPVs time (sec.)
1	51	361	50.37	38.84
2	128	515	50.37	36.25
3	300	1029	50.37	37.20
4	572	2088	50.37	41.84
5	912	3723	50.37	58.62
6	1231	5559	50.37	80.91
7	1516	7460	50.37	185.58
8	1718	9006	50.35	286.18
9	1850	10135	50.26	388.35
10	1950	11095	50.26	623.57
20	2114	13130	50.14	736.97
30	2125	13385	50.10	936.65
40	2125	13385	50.10	934.90
50	2125	13385	50.10	1013.60

Table 6.10: The result of solving instance 2 over the 17-attribute database using model IPVs for different values of s.

s	Number of views in $\lvert V_s'' \rvert$	Number of indexes in IPVs	IPVs cost	IPVs time (sec.)
1	51	313	50.97	23.08
2	99	409	50.97	22.27
3	272	885	50.97	23.23
4	616	2144	50.97	35.91
5	1071	4221	50.95	82.40
6	1489	6520	50.92	183.43
7	1802	8534	50.87	428.71
8	2005	10064	50.87	2502.35
9	2124	11100	–	>1hr
10	2184	11696	–	>1hr
20	2210	13287	–	>1hr
30	2321	13540	–	>1hr
40	2321	13540	–	>1hr
50	2321	13540	–	>1hr

Table 6.11: The result of solving instance 3 over the 17-attribute database using model IPVs for different values of s.

s	Number of views in $\lvert V_s'' \rvert$	Number of indexes in IPVs	IPVs cost	IPVs time (sec.)
1	51	375	50.62	14.87
2	77	427	50.62	14.14
3	152	622	50.62	14.58
4	292	1077	50.62	15.62
5	511	1904	50.62	19.36
6	789	3099	50.62	25.36
7	1085	4512	50.23	28.59
8	1390	6117	50.23	30.51
9	1666	7705	50.23	52.40
10	1928	9366	50.23	132.73
20	2953	20717	50.21	751.72
30	2981	21383	50.15	984.80
40	2982	21415	50.15	800.30
50	2982	21415	50.15	799.18

Table 6.12: The result of solving instance 4 over the 17-attribute database using model IPVs for different values of s.

s	Number of views in $\lvert V_s'' \rvert$	Number of indexes in IPVs	IPVs cost	IPVs time (sec.)
1	51	391	50.12	12.69
2	98	485	50.12	11.67
3	253	917	50.12	11.86
4	599	2159	50.12	18.76
5	1141	4539	50.12	28.52
6	1790	7875	50.12	79.78
7	2436	11678	50.12	226.64
8	2978	15291	50.12	497.48
9	3420	18624	50.12	971.89
10	3743	21324	50.12	1296.46
20	4529	30595	–	>1hr
30	4556	31247	–	>1hr
40	4556	31247	–	>1hr
50	4556	31247	–	>1hr

Table 6.13: Comparing model IPVs when $s = 1$ and model IPN for 18 instances over a 13-attribute database. Instances are the same as instances in Table 6.6.

instance number	number of queries	IPVs ($s = 1$) cost	IPN cost	IPVs ($s = 1$) time (sec.)	IPN time (sec.)
1	50	50.06	50.04	12.43	101.53
2	50	50.07	50.03	8.88	108.64
3	50	50.06	50.04	8.87	66.62
4	60	60.27	60.08	2.14	89.99
5	60	60.62	60.09	1.94	83.92
6	60	60.20	60.02	2.04	61.88
7	70	71.95	70.17	2.89	384.35
8	70	70.32	70.10	2.39	304.88
9	70	70.43	70.09	2.44	158.98
10	80	80.29	80.10	5.26	277.67
11	80	80.74	80.04	4.33	423.09
12	80	80.95	80.25	3.79	308.7
13	90	90.37	90.07	5.77	282.77
14	90	90.16	90.05	4.99	449.66
15	90	90.28	90.05	4.02	505.76
16	100	100.32	100.05	6.92	287.08
17	100	100.13	100.06	5.56	485.64
18	100	101.12	100.11	6.61	667.93

all of these instances, the value of cost obtained by model IPVs with s=1 is very close to the value of cost obtained from model IPN (within at most 2%). Also, in all of these instances we observe that the time required to solve each instance using model IPVs was significantly shorter than the time required to solve that instance using model IPN, on average 66 times.

We also compare the value of cost obtained from model IPVs ($s = 1$) to the value of cost obtained from model IPNIRp with $p = p_{0.01}$ for the eighteen instances of Table 6.13. Also, we compare the total time required to solve each instance using model IPVs ($s = 1$) and the total time required to solve the corresponding model IPNIRp with $p = p_{0.01}$. We present the results in Table 6.14. From this table we observe that the quality of solutions obtained from model IPVs ($s = 1$) is not significantly

Table 6.14: Comparing model IPVs when $s = 1$ and model IPNIRp with $p = p_{0.01}$ for 18 instances over a 13-attribute database. Instances are the same as instances in Table 6.6.

instance number	number of queries	IPVs ($s = 1$) cost	IPNIRp with $p = p_{0.01}$ cost	IPVs ($s = 1$) time (sec.)	IPNIRp with $p = p_{0.01}$ time (sec.)
1	50	50.06	50.07	12.43	10.55
2	50	50.07	50.09	8.88	16.09
3	50	50.06	50.29	8.87	8.42
4	60	60.27	60.30	2.14	10.32
5	60	60.62	60.22	1.94	47.74
6	60	60.20	60.32	2.04	5.35
7	70	71.95	70.28	2.89	66.48
8	70	70.32	70.79	2.39	7.75
9	70	70.43	70.44	2.44	7.12
10	80	80.29	80.67	5.26	12.74
11	80	80.74	80.21	4.33	52.39
12	80	80.95	80.47	3.79	14.27
13	90	90.37	90.67	5.77	14.10
14	90	90.16	90.45	4.99	23.18
15	90	90.28	90.49	4.02	17.06
16	100	100.32	100.85	6.92	43.06
17	100	100.13	100.92	5.56	20.99
18	100	101.12	100.51	6.61	28.52

different from the quality of solutions obtained from model IPNIRp with $p = p_{0.01}$, however, the amount of time needed to solve each instance using IPVs ($s = 1$) is on average 6 times less than the time required to solve the corresponding model IPNIRp with $p = p_{0.01}$.

We changed the value of α (defined in the context of formula 4.1) in some of the above instances (instances 1, 4, 7, 10, 13, and 16) and solved these instances using model IPVs with s=1. Tables 6.15 and 6.16 presents the value of cost obtained by models IPN and IPVs (s=1) as well as the time required to solve each of these models for $\alpha = 0.1$ and $\alpha = 0.2$, respectively. From these two tables we observe that except

Table 6.15: Comparing model IPVs when $s = 1$ and model IPN for some of the instances in Table 6.13 when $\alpha = 0.1$.

instance number	number of queries	IPVs ($s = 1$) cost	IPN cost	IPVs ($s = 1$) time (sec.)	IPN time (sec.)
1	50	141783.00	103654.00	45.86	934.90
4	60	402.90	402.90	2.18	40.25
7	70	683.97	683.97	2.84	124.27
10	80	182.84	182.84	3.88	149.56
13	90	243.15	243.15	5.31	299.53
16	100	100.32	100.05	7.13	418.76

Table 6.16: Comparing model IPVs when $s = 1$ and model IPN for some of the instances in Table 6.13 when $\alpha = 0.2$.

instance number	number of queries	IPVs ($s = 1$) cost	IPN cost	IPVs ($s = 1$) time (sec.)	IPN time (sec.)
1	50	107.71	107.71	8.23	75.50
4	60	67.89	66.31	2.05	330.91
7	70	86.98	85.78	2.49	398.71
10	80	83.47	82.15	5.90	1042.91
13	90	92.88	91.83	6.10	524.91
16	100	104.26	102.82	6.97	498.39

for instance 1 in Table 6.15, the value of cost obtained by model IPVs with s=1 is very close to the value of cost obtained from model IPN (within 2%). For instance 1 in Table 6.15, when we increased the value of s to 3, the value of cost obtained by model IPVs lies within 2% of the value cost obtained by model IPN. Also, in all of these instances we observe that the time required to solve model IPVs ($s = 1$) is significantly shorter than the time required to solve the corresponding model IPN: on average 39 times for instances in Table 6.15 and 110 times for instances in Table 6.16.

In our experiments with model IPVs, we observe that $s = 1$ provides solutions which are close to the solutions obtained from model IPN. We investigated the reason for this phenomenon. We observed that the size of 94% of the views in the 13-attribute database are either less than 10% of the size of the raw-data view or more than 90%

of the size of the raw-data view, i.e., views are either very small or very large. As a result for small queries like query q, a view v which is the same as query q is selected. Larger queries like q' are answered by the raw-data view $v1$ where $size(q) \approx size(v1)$. Thus when $s = 1$, i.e., when the search space of views is limited to only those views that are the same as queries and the raw-data view, we still get good solutions from Model IPVs. But this phenomenon doesn't necessarily happen when the storage space bound is small, as we observed in instance 1 of Table 6.15. Note that in this instance $\alpha = 0.1$ and storage space is very limited.

In general, for each instance it seems reasonable to increase the value of s as much as possible while the corresponding execution time remains relatively small.

6.4 Comparison with the Heuristic Approach ACN

In this section, we compare the search space of views and indexes in model IPN with the search space of views and indexes generated by algorithm ACN which is an OLAP modified approach proposed in [2]. The approach for selecting views and indexes explained in [2] is as follows:

- First reduce the size of the search space of views by a "view-merging" approach.
- Next use a Greedy(n, k) algorithm[3] to reduce the size of the search space of indexes.
- Finally use another Greedy(n, k) algorithm to find a set of views and indexes simultaneously among the reduced search spaces of views and indexes.

The authors in [2] do not define the cost of answering queries and just mention that the cost is calculated by an estimation module.

We used the approach of [2] to develop algorithm ACN as follows: We reduce the size of the search space of views and indexes in the same way as in [2]. However, instead of applying a Greedy(n, k), we use our integer programming model to find the

[3]In a Greedy(n, k) algorithm, first each combination of n indexes are evaluated and the combination which result in the lowest cost would be selected. Then from the $k - n$ indexes left, indexes are selected one by one in a greedy manner until no more indexes can be selected due to the space constraint.

best set of views and indexes among the reduced search space of views and indexes. This way we guarantee that we find the best set of views and indexes in the reduced search space. Also, we use our cost model defined in Section 2.1, rather than the estimation module in [2].

In the first step of algorithm ACN, each query is considered to be a potential view, and a randomly selected order of the attributes of each such view is considered an index for that view. Subsequently, ACN considers the union of each pair of views, v_1 and v_2, as a new "merged" view v. If the size of a merged view v is less than the sum of the sizes of views v_1 and v_2, the algorithm adds view v to the search space of views. In this case, for each index of views v_1 and v_2, we also construct a similar index for the merged view v: If π is an index of view v_1, we add the attributes in $v \setminus v_1$ to the end of the sequence of attributes of index π to get an index for view v. Similarly, we get the other indexes of v from indexes of $v2$. ACN terminates once it can add no more merged views to the search space.

In each instance of the OLAP-VI problem, suppose V^{ACN} is the set of views generated by algorithm ACN and $\Pi^{ACN}(v)$ is the set of indexes for each view $v \in V^{ACN}$. We define model IPACN as a model which is the same as model IPN, except that we use the set of views in V^{ACN} in place of V' and the set of indexes $\Pi^{ACN}(v)$ in place of $\Pi''(v)$ for each view $v \in V^{ACN}$.

Our goals in this section is to (1) compare the size of the search space of views and indexes in our proposed model IPN with the size of the search space of views and indexes generated by algorithm ACN, and (2) compare the quality of the search spaces generated through these two approaches in terms of containing a set of views and indexes which result in lower cost of answering the queries.

Our first set of experiments consists of ten instances over the 17-attribute database. Each instance has twenty queries in the workload. For each instance we measure the size of the search space of views and indexes in model IPN and the size of the search space of views and indexes generated by algorithm ACN. Also, for each instance we solve models IPN and IPACN. In Figure 6.1(a) we compare the size of the search spaces of views of IPN and IPACN, and in Figure 6.1(b) we compare the size of the

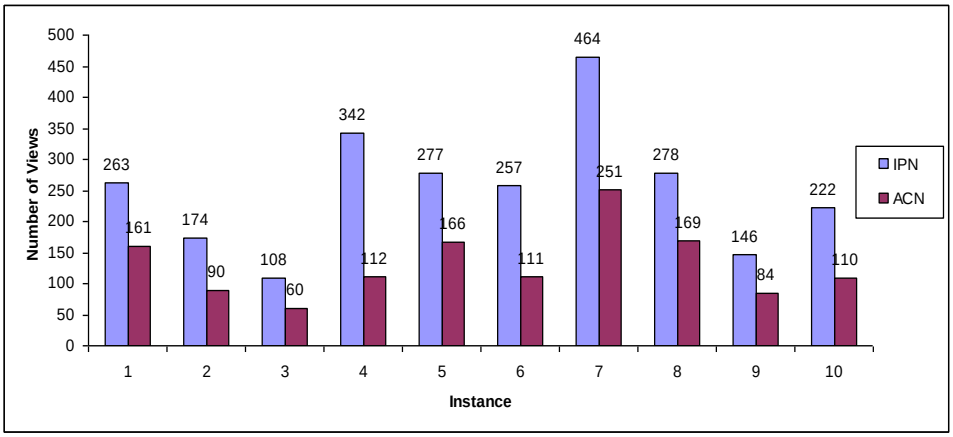
Table 6.17: Comparison of (1) the costs obtained from solving models IPN and IPACN, and (2) the time required to solve each of these instances using models IPN and IPACN. Instances are over the 17-attribute database. Each instance has 20 queries.

instance	IPN cost	IPACN cost	IPN time (sec.)	IPACN time (sec.)
1	20.84	359360.00	12.08	11.95
2	20.00	243476.70	1.22	10.43
3	20.68	92580.19	17.51	7.98
4	20.20	1810970.47	2.58	16.06
5	22.08	576012.00	2.44	32.20
6	20.12	144209.00	3.99	5.43
7	21.42	743439.54	11.02	40.76
8	20.00	35627.84	38.63	12.72
9	20.51	481878.12	6.94	4.28
10	20.14	43369.08	33.51	4.45

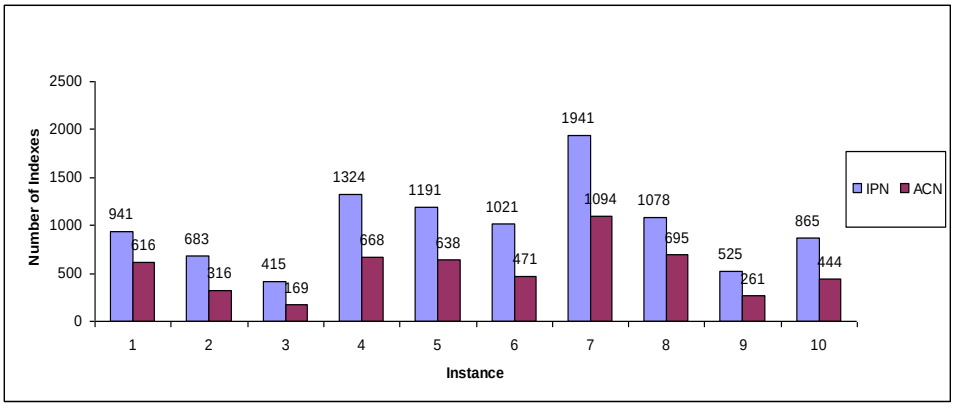
search spaces of indexes of IPN and ACN for each instance. For all of these instances, the search space of views and indexes generated by ACN is significantly smaller than the search space of views and indexes in IPN. On average, both the number of indexes and the number of views in IPACN is half the number of indexes and the number of views in IPN.

For each of these instances, we also measure the cost obtained from solving models IPN and IPACN and report them in Table 6.17. In this table, we also report the total time required to solve each instance using model IPN and the time required to solve the corresponding model IPACN. From this table we observe that for each instance, the cost obtained using model IPN is significantly smaller than the cost obtained using model IPACN. Also, we observe that the total time of solving each instance using model IPN is comparable with the total time of solving that instance using model IPACN.

Our second set of experiments consists of ten instances over the 13-attribute database. Each instance has twenty queries. Again, for each instance we compare the size of the search space of views and indexes in model IPN and IPACN and present



(a) Comparison of the sizes of the search spaces of views.



(b) Comparison of the sizes of the search spaces of indexes .

Figure 6.1: Comparison of the size of the search space of views and indexes in models IPN and IPACN for the instances over the 17-attribute database.

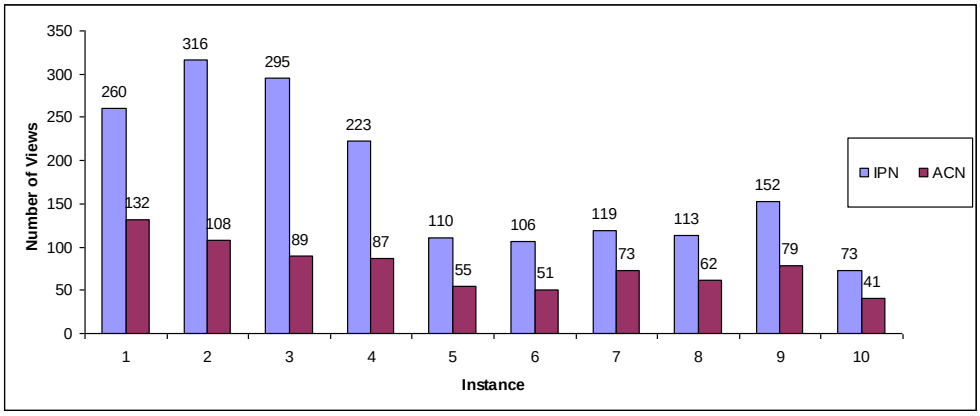
Table 6.18: Comparison of (1) the costs obtained from solving models IPN and IPACN, and (2) the time required to solve each of these instances using models IPN and IPACN. Instances are over the 13-attribute database. Each instance has 20 queries.

instance	IPN cost	IPACN cost	IPN time (sec.)	IPACN time (sec.)
1	85.16	469959.42	28.59	7.72
2	25.42	769250.00	1.88	37.19
3	21.10	1237010.15	4.57	21.52
4	20.66	656023.71	5.18	13.08
5	20.23	598928.00	2.13	27.4
6	20.03	74779.27	4.17	5.6
7	20.19	72352.68	1.31	8.47
8	20.01	19959.04	1.8	15.58
9	20.01	336517.00	15.16	19.03
10	20.12	3839.23	3.4	3.98

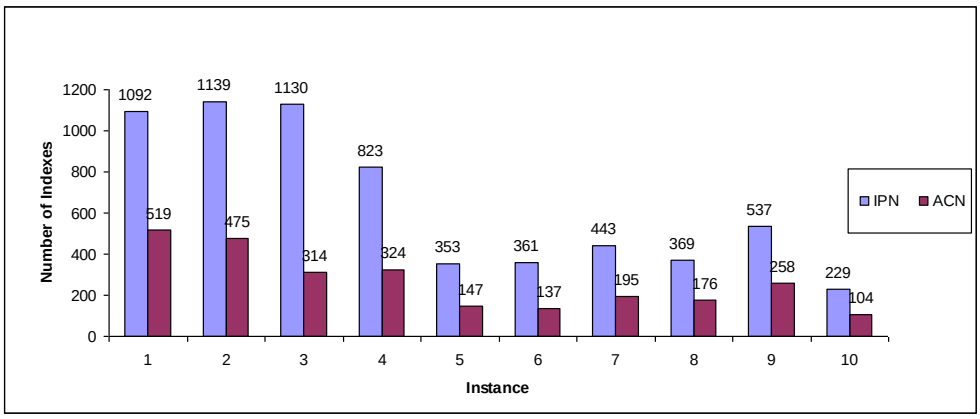
the results in Figures 6.2(a) and 6.2(b). For all of these instances, we observed that the search space of views and indexes generated by ACN is smaller than the size of the search space of views and indexes in IPN. On average, both the number of indexes and the number of views in ACN is half of the number of indexes and the number of views in IPN.

For each of these instances, we also measured the values of cost obtained from solving models IPN and IPACN and report them in Table 6.18. In this table, we also report the total time required to solve each instance using model IPN and the time required to solve that instance using the corresponding model IPACN. Consistent with our previous set of experiments, from Table 6.18 we observe that for each instance, the cost obtained using model IPN is significantly smaller than the cost obtained using model IPACN. Also, we observe that the total time of solving each instance using model IPN is comparable with the total time of solving that instance using model IPACN.

Discussion. We believe the reason for the relatively poor performance of algorithm ACN is the manner in which its indexes are selected. In fact, it is very likely



(a) Comparison of the sizes of the search spaces of views.



(b) Comparison of the sizes of the search spaces of indexes .

Figure 6.2: Comparison of the size of the search space of views and indexes in model IPN and IPACN for the instances over the 13-attribute database.

that each index in the search space of indexes generated by Algorithm ACN is useful for one or only a few of the corresponding queries. Since the storage space is not enough to select one distinct index per query, there may be no useful index among the selected indexes for some of the queries. To clarify this, we present the following example:

Example 6. Suppose queries in the workload are q_1=$\{a, b\}$ and q_2=$\{a, c\}$. Suppose the sizes of views v_a=$\{a\}$, v_b=$\{b\}$, v_c=$\{c\}$, v_1=$\{a, b\}$, v_2=$\{a, c\}$, and v= $\{a, b, c\}$ are 10000, 2000, 15000, 15000, 17000, and 20000, respectively. In the first iteration of ACN, v_1 and v_2 are in the set of potential views. Suppose the order of the attributes of the index for v_1 is (b, a) (a random order of attributes of q_1), and the order of attributes of the index for v_2 is (c, a) (a random order of attributes of q_2). Since $s(v) \leq s(v1) + s(v2)$ (i.e., $20000 \leq 32000$), in the second iteration view v is added to the set of potential views, and the set of potential indexes for v would be $\{(b, a, c), (c, a, b)\}$. Suppose the space limit is 40000. Thus the optimal solution of model IPACN for this instance includes view v and index (c, a, b), at the cost of 20001.18. However, in the optimal solution (obtained from model IP2), view v and index (a, c, b) over view v would be selected, and the value of cost in the optimal solution is 3.18.□

6.5 Comparing with the Heuristic Approach GHRU

In all procedures that we proposed in this dissertation, we used integer programming (IP) models to find the best set of views and indexes among those considered. In this section, we compare our IP-based approach with a greedy heuristic approach proposed in [13] for selecting views and indexes. We refer to this algorithm as algorithm GHRU.

Algorithm GHRU

There are two algorithms proposed in [13] for selecting views and indexes. For our experimental comparisons we chose, from the algorithms proposed in [13], the algorithm with the best performance guarantees, i.e., the r-greedy algorithm, and set $r = 4$ for the parameter of this algorithm as suggested in this paper. GHRU includes two types of basic steps: (1) select an index for an already selected view; or (2) pick a view and at most three indexes. In each iteration, GHRU selects an index or a combination of a view with at most three indexes that maximizes the benefit per unit space by enumeration. For a full description of the algorithm see [13].

We apply algorithm GHRU on several different search spaces of views and indexes that we proposed in Chapters 3 and 4, and evaluate the quality of the solutions obtained from this algorithm. More specifically, we apply GHRU on the original search space of views and indexes, the search space of views and indexes in model IP2, and the search space of views and indexes in model IPN. Also, we apply GHRU on the search space of views and indexes generated by algorithm ACN which we explained in the previous section.[4]

Our first set of experiments include six instances over the 7-attribute database. We used the smallest size database so that we will be able to apply GHRU on the original search space.[5] For each instance, we measured the cost obtained via GHRU on each search space and report the results in Table 6.19. Also, we measured the total time required to build each search space and apply GHRU on that search space, and report it in Table 6.20. From Table 6.19 we observe that the quality of the solutions obtained via GHRU when it is applied on the search space of IPN is significantly better than the quality of the solutions obtained when it is applied on the other search spaces. Furthermore, from Table 6.20 we observe that on average the time required to solve instances by applying GHRU on the original, IP2, IPN, and ACN

[4]Algorithm GHRU in [13] is primarily designed to be applied on the original search space of views and indexes.

[5]Note that the size of the original search space of views and indexes is huge for larger databases; thus GHRU cannot be applied (within reasonable execution time limits) on the original search space of views and indexes for instances on larger databases.

Table 6.19: The value of cost obtained from GHRU when it is applied on different search spaces for 6 instances over the 7-attribute database.

instance number	search space			
	original	IP2	IPN	ACN
1	1173610	15	16	1396639
2	1199264	299866	299866	900263
3	900710	68	44	127
4	1199262	899451	43	1199288
5	1199262	299826	13	1061397
6	1199262	599754	299853	594775

search spaces are 615.50, 12.63, <0.01, and 0.02 seconds, respectively. Thus, not only we obtained the best (smallest) cost when we applied GHRU on the IPN search space, but also applying GHRU on the IPN was faster than applying GHRU on any other search space. This implies that most of the time GHRU by itself does not select a good combination of views and indexes, however, if the search space is reduced by our approaches prior to applying GHRU, then GHRU selects a relatively better combination of views and indexes. Also, GHRU did not provide good solutions when it was applied on ACN search space. We believe this is due to the fact that for most instances the ACN search space does not contain good combination of views and indexes.

In the second set of experiments in this section, we compare the performance of algorithm GHRU with our inexact approach IPN for a set of six instances over a 7-attribute database where each instance has ten queries. The results are displayed in Table 6.21. For each instance, we report (1) the value of cost obtained via model IPN, (2) the cost obtained using algorithm GHRU, and (3) the corresponding execution times. From this table we observe that:

- In all six instances, the cost obtained using model IPN is significantly lower than the corresponding cost we obtained using algorithm GHRU. See Remark 1 below.

Table 6.20: The total execution time of applying GHRU on different search spaces for 6 instances over the 7-attribute database. (In these instances, the time required to find each search space is significantly shorter than the time required to apply GHRU on that search space.)

instance number	search space			
	original	IP2	IPN	ACN
1	817.78	38.13	<0.01	<0.01
2	524.42	17.33	<0.01	0.03
3	614.86	15.07	<0.01	0.04
4	641.01	0.01	<0.01	<0.01
5	571.10	5.25	<0.01	<0.01
6	523.84	0.01	<0.01	0.04

Table 6.21: Comparison of (1) the cost obtained from solving model IPN with the cost obtained by algorithm GHRU, and (2) the execution times of solving IPN with the required times to apply algorithm GHRU for each of the instances over a 7-attribute database. Each instance has 10 queries.

instance	cost		execution time (sec.)	
	IPN	GHRU	IPN	GHRU
1	12.52	1,173,633.70	0.93	817.85
2	53.96	1,199,286.60	0.61	524.49
3	31.59	900,728.16	0.63	615.02
4	11.68	1,199,262.84	0.60	641.09
5	12.15	1,473,421.00	0.60	571.19
6	14.85	1,199,290.98	0.60	523.90

- The execution time for solving each instance using GHRU is significantly higher than the time required to solve model IPN for the corresponding instance. See Remark 2 below.

Remark 1. One can easily observe that there is a large difference between the costs obtained by the two methods, with the cost obtained by IPN model being consistently much lower than that obtained via GHRU. This large difference is partly due to the special structure of the instances in our experiments. Note that by the assumptions made earlier for every instance in our experiments all attributes of each query are in its `WHERE` clause. Hence the corresponding cost of answering this query using an appropriate view and a proper index is as low as 1. It follows that the optimal cost of answering a given collection of queries could be as low as the number of queries. For this collection of instances model IPN consistently finds a cost relatively close to this lower bound. This implies that the corresponding solutions contain a proper mixture of views and indexes so that each query is answered by an appropriate view and an associated index. On the other hand, if query q is answered by view v with no index on v, then the cost of answering this query is equal to the size of the view itself. In the database that we used for these instances, i.e., the 7-attribute database, the average size of a view is 216,469 rows, and in the solution obtained by algorithm GHRU there are many cases where we have no useful index for answering a query. This explains the huge difference between the costs obtained using the two approaches. If the structure of the instance allows some attributes of a query to be in its GROUP BY clause, then we expect the difference between the costs obtained by the two approaches to be somewhat more moderate, although still potentially significant.

Remark 2. In fairness, we must add that algorithm GHRU was originally proposed to solve the problem where the collection of queries is relatively large (e.g., the entire collection of possible queries). For each such instance the execution time for solving the corresponding model IPN may be prohibitively excessive, especially if the number of attributes in the database is relatively large, but algorithm GHRU may terminate within more reasonable time limits.

6.6 Summary of Observations

Following is a summary of our observations regarding the inexact (heuristic) methods that we considered for solving the OLAP-VI problem.

- We are able to solve large realistic-size instances of the problem using the inexact model IPN, while we cannot solve these instances using the exact model IP2 due to the excessive execution time or computer memory shortage. In all instances in our experiments where we know the optimal value of cost, the value of cost obtained from model IPN is either optimal or close to optimal (within 9% of the optimal).

- For all instances that we were able to solve model IPN, we obtained a solution from model IPNIRp with $p = 10$ where its value of cost was within 1% of the value of cost obtained from model IPN. Also, for each of these instances we could solve IPNIRp with $p = 10$ in a significantly shorter amount of time compared to IPN.

- In our experiments, most of the times model IPVs provides solutions of acceptable qualities when $s = 1$.

- The search space of views and indexes of model IPN contains a feasible set of views and indexes for the OLAP-VI problem that is significantly better than the best feasible set of views and indexes in the search space generated by algorithm ACN (derived from the approach proposed [2]) in terms of reducing the total cost of answering the queries.

- When we apply the algorithm of [13] (GHRU) on the search space of views and indexes of model IPN, i.e., V' and $\Pi''(v)$ for each $v \in V'$, we observed that this algorithm performs significantly better compared to the case when it is applied on the original search space of views and indexes, i.e., V and $\Pi(v)$ for each $v \in V$. In other words, the GHRU algorithm of [13] by itself does not select a good combination of views and indexes. However, if the search space is reduced

through our inexact approach of section 5.1, then this algorithm selects a much better combination of views and indexes, albeit still significantly larger than the corresponding optimal value.

- In our experiments we observed that not only our inexact approach IPN provides significantly better solutions that algorithm GHRU (in terms of reducing the total cost of answering the queries) , but it also runs significantly faster than GHRU.

Chapter 7

Conclusions and Future Research

In this dissertation we undertook a systematic study of the OLAP view- and index-selection problem under the storage space constraint. The input of the view- and index-selection problem includes a data warehouse schema, a set of data-analysis queries of interest, and an upper bound b on the available storage space, and the output is a collection of views and indexes that would fit within the storage limit b and would minimize the cost measure (evaluation time) for the given queries. We proposed several exact and inexact methods to solve this problem. Our specific contributions to the view- and index-selection problem are as follows.

- We introduced an integer programming model for the OLAP-VI problem.
- We developed several algorithms that effectively and efficiently prunes the space of potentially beneficial views and indexes, and provided formal proofs that our pruning algorithms keep at least one globally optimal solution in the search space, thus the resulting integer-programming model is guaranteed to find an optimal solution. Using our approach, we could solve moderate-size realistic instances of the OLAP-VI problem optimally.
- We developed several algorithms to further reduce the size of the search space of

views and indexes using the structure of the OLAP-VI problem so that we are able to solve larger instances of the problem, although we no longer guarantee the global optimality of the resulting solution; and

- we presented an experimental evaluation of our algorithms and compared our approaches with the well-known approaches in [2, 13].

  Our experiments show that our proposed approaches to view and index selection result in high-quality solutions — in fact, in *globally optimal* solutions for many realistic-size problem instances. Thus, they compare favorably with the well-known OLAP-centered approach of [13] and provide for a winning combination with the end-to-end framework of [2] for generic view and index selection.

Our contributions open new avenues for view and index selection and materialization in OLAP and other systems. Specifically, this research lays the foundation for studying this and other versions of the view- and index-selection problem in a systematic principled way. In addition, our contributions make it possible, in *practical* settings, to quantify the "goodness" of specific view- and index-selection solutions with respect to the best possible (that is, *globally optimal*) counterparts, rather than just with respect to the base line where the system does not use any views. Finally, in a broader context [1] it has become clear that advances in view or index selection and advances in query rewriting using views and indexes are interrelated, thus these problems need to be studied together.

7.1 Future Research Avenues

We envision two immediate directions to extend the approaches outlined in this dissertation. One direction of research pertains to considering attribute preferences for queries and select indexes based on these preferences. The other direction is to consider sparse indexes in the search space of indexes:

Considering attribute preferences

In practice, sometimes it is more important to have an index on a given subset of the attributes of a query, rather than on all of its attributes. Thus, a possible future research is to consider different effects of indexes in answering the queries that is not limited to the *order* of the attributes of indexes.

Including sparse indexes

In our research we limited the choice of indexes to the fat indexes, i.e., those indexes that include all attributes of their corresponding view. In practice however, sparse indexes can be very beneficial to answer queries. In fact, since in our OLAP-VI problem, storage space is the main constraint, by allowing sparse indexes in the search space we may be able to use the available storage space more efficiently. As a result, another possible future research is to consider sparse indexes in the search space. However, by adding sparse indexes, the size of the search space of indexes and the complexity of the problem would increase dramatically even for relatively small instances.

Bibliography

[1] F. N. Afrati and R. Chirkova. Selecting and using views to compute aggregate queries (extended abstract). In *Proceedings of the 10th International Conference on Database Theory*, pages 383–397, 2005.

[2] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 496–505, 2000.

[3] Z. Asgharzadeh Talebi, R. Chirkova, and Y. Fathi. Exact and inexact methods for solving the problem of view selection. *International Journal of Business Intelligence and Data Mining*, 4(3/4):391– 415, 2009.

[4] Z. Asgharzadeh Talebi, R. Chirkova, Y. Fathi, and M. Stallmann. Exact and inexact methods for selecting views and indexes for OLAP performance improvement. In *Proceedings of the 11th International Conference on Extending Database Technology*, pages 311–322, 2008.

[5] A. Balmin, F. Ozcan, K. Beyer, R. Cochrane, and H. Pirahesh. A framework for using materialized xpath views in xml query processing. In *Proceedings of the 30th International Conference on Very Large Data Bases*, pages 60–71, 2004.

[6] E. Baralis, S. Paraboschi, and E. Teniente. Materialized view selection in a multidimensional database. In *Proceedings of the 23th International Conference on Very Large Data Bases*, pages 156–165, 1997.

[7] C. M. Broughton. IBM DB2 cube views and DB2 materialized query tables in a SAS environment. `http://www.sas.com/partners/directory/ibm/cubeviews.pdf`, 2005.

[8] A. Caprara, M. Fischetti, and D. Maio. Exact and approximate algorithms for the index selection problem in physical database design. *IEEE Transactions on Knowledge and Data Engineering*, 7(6):955–967, 1995.

[9] S. Chaudhuri, M. Datar, and V. R. Narasayya. Index selection for databases: A hardness study and a principled heuristic solution. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1313–1323, 2004.

[10] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74, 1997.

[11] S. Chaudhuri and V. R. Narasayya. An efficient cost-driven index selection tool for microsoft SQL server. In *Proceedings of the 23th International Conference on Very Large Data Bases*, 1997.

[12] C. I. Ezeife. A uniform approach for selecting views and indexes in a data warehouse. In *Proceedings of the 1997 International Symposium on Database Engineering and Applications*, pages 151–160, 1997.

[13] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *Proceedings of the 13th International Conference on Data Engineering*, pages 208–219, 1997.

[14] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 205–216, 1996.

[15] ILOG. CPLEX Homepage, 2004. Information on CPLEX is available at `http://www.ilog.com/products/cplex/`.

[16] P. Kalnis, N. Mamoulis, and D. Papadias. View selection using randomized search. *Data Knowledge Engineering*, 42(1), 2002.

[17] H. J. Karloff and M. Mihail. On the complexity of the view-selection problem. In *Proceedings of the 18th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 167–173, 1999.

[18] R. Kimball and M. Ross. *The Data Warehouse Toolkit (second edition).* Wiley Computer Publishing, 2002.

[19] J. Kratica, I. Ljubic, and D. Tosic. A genetic algorithm for the index selection problem. *Applications of Evolutionary Computing*, 2611:281–291, 2003.

[20] L. L. and L. Wolsey. *Integer Programming.* Wiley, USA, 1998.

[21] J. Li, Z. Asgharzadeh Talebi, R. Chirkova, and Y. Fathi. A formal model for the problem of view selection for aggregate queries. In *Advances in Databases and Information Systems, 9th East European Conference, Tallinn, Estonia*, pages 125–138, 2005.

[22] Microsoft. Web page of the AutoAdmin project: Self-tuning and self-administering databases. `http://research.microsoft.com/research/dmx/autoadmin`.

[23] Microsoft. Web page of the data management, exploration and mining group. `http://research. microsoft.com/research/dmx/`.

[24] A. Shukla, P. Deshpande, and J. F. Naughton. Materialized view selection for multidimensional datasets. In *Proceedings of 24rd International Conference on Very Large Data Bases*, pages 488–499, 1998.

[25] Transactions Performance Processing Council. TPC Benchmark-H Statndard Specification Revision 2.1.0. `http://www.tpc.org/tpch/spec/tpch2.1.0.pdf`, 2002.

[26] G. Valentin, M. Zuliani, D. Zilio, G. Lohman, and A. Skelley. DB2 advisor: An optimizer smart enough to recommend its own indexes. In *Proceedings of the 16th International Conference on Data Engineering*, 2000.

[27] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 136–145, 1997.

[28] D. Zilio, C. Zuzarte, S. Lightstone, L. G. Ma, W., R. Cochrane, H. Pirahesh, L. Colby, J. Gryz, E. Alton, D. Liang, and G. Valentin. Recommending materialized views and indexes with IBM DB2 design advisor. In *Proceedings of the 16th International Conference on Data Engineering*, 2004.

Appendix

Appendix A

An Algorithm to Build Digraph G_v for View v

Following we provide a detailed pseudocode (based on our C++ code) for our algorithm to build digraph G_v for view v which is a view in a view lattice. The inputs of the algorithm are the set $Q(v) = \{q_1, q_2, \ldots, q_g\}$ and v. The first step of the algorithm is to build the set of nodes of G_v which is array L in the following code. Note that each element of L is a set of attributes.

```
for i=1:g do
  list[i]={qi}
  L[i]=qi
s1=0
s2=g
#nodes=g
while s1!=s2
  for i=1:g do
    for j=s1+1:s2 do
      if qi intersect L[j] is not equal to the empty set
      and qi is not an element of list[j]
        #nodes = #nodes + 1
```

```
        L[#nodes] = qi intersect L[j]
        list[#nodes] = list[#nodes] union {qi}
      endif
    endfor
  endfor
  s1 = s2
  s2 = #nodes
endwhile
if v is not equal to any L[i] (1 <= i <= #nodes)
  #nodes = #nodes + 1
  L[#nodes]=v
endif
remove repetitive nodes and update #nodes
```

The next part of the code is to build the adjacency list for graph G:

```
for i=1:#nodes
  for j=i+1:#nodes
    if L[i] is a subset of L[j]
      t=true
      k=1
      while t=true and k<=#nodes
        if k!=i and k!=j and L[i] is a subset of L[k] and
        L[k] is a subset of L[j] then t=false
        k=k+1
      endwhile
      if t=true
        add L[j] to the adjacency list of L[i]
      endif
    endif
    if L[j] is a subset of L[i]
      t=true
      k=1
      while t=true and k<=#nodes
        if k!=i and k!=j and L[j] is a subset of L[k] and
        L[k] is a subset of L[i] then t=false
        k=k+1
      endwhile
      if t=true
       add L[i] to the adjacency list of L[j]
      endif
```

```
    endif
  endfor
endfor
```

Appendix B

An Algorithm to Find the Elements of the Set $\Pi'(v)$ for view v

Following we provide a detailed pseudocode (based on our C++ code) for our algorithm to find the elements of the set $\Pi'(v)$ for view v which is a view in a view lattice. The input of this algorithm is the digraph G_v.

```
numOfPaths[{}]=1;
path[{}][1]=({})
for all nodes u in Gv except for the source node (i.e., empty set)
  numOfPaths[u]=0
end for
for each node v in Gv in topological order
  for i=1:numOfPaths[v]
    for each parent node u of node v
      numOfPaths[u]=numOfPaths[u]+1
      path[u][numOfPaths[u]]=path[v][i],u
    end for
    remove path[v][i]
  end for
end for
```

```
construct the index associated with each of the remaining paths
```

Appendix C

An Example for Constructing the Set $\Pi''(v)$ for a Given View v

In this appendix, we provide an example to show how we can apply Algorithm 1 in Chapter 5 to find the elements of the set $\Pi''(v)$ for a given view v.

Example 7. Consider view $v = \{a, b, c, d\}$ and the set of queries $Q(v) = \{q_1, q_2, q_3\}$ where $q_1 = \{a, b\}$, $q_2 = \{a, c\}$, and $q_3 = \{b, d\}$. Let $size(\{a\}) = 200$, $size(\{b\}) = 100$, $size(\{a, b\}) = 250$, $size(\{a, c\}) = 400$, $size(\{b, d\}) = 200$, and $size(\{a, b, c, d\}) = 1000$. The corresponding digraph G_v is presented in Figure C. We have N_v=$|Q(v)|$=3. Following we apply our algorithm to find the first element of $\Pi''(v)$. The other two indexes of $\Pi''(v)$ can be found similarly:

$MCS(q_1) = MCS(q_2) = MCS(q_3) = MCS(q_4) = s(v) = 1000$

$cost(\emptyset, q_1) = cost(\emptyset, q_2) = cost(\emptyset, q_3) = cost(\emptyset, q_4) = s(v) = 1000$

$Q_{temp} = \{q_1, q_2, q_3, q_4\}$

$r \leftarrow 1 \leq N_v = 3$

$w = \{a\}$

$Q' = \emptyset$

$u = \emptyset$

$cost(u) = 0$

$\hat{u} = \emptyset$

$perm(\{a\}) = (a)$

$cost((a), q_1) = \frac{1000}{200} = 5$

$cost((a), q_2) = \frac{1000}{200} = 5$

$cost((a), q_3) = 1000$

$w = \{b\}$

$Q' = \emptyset$

$u = \emptyset$

$cost(u) = 0$

$\hat{u} = \emptyset$

$perm(\{b\}) = (b)$

$cost((b), q_1) = 1000$

$cost((b), q_2) = \frac{1000}{100} = 10$

$cost((b), q_3) = \frac{1000}{100} = 10$

$w = \{a, c\}$

$Q' = \{q_2\}$

$u = \{a\}$

$cost(u) = 5$

$\hat{u} = \{a\}$

$perm(\{a, c\}) = (a, c)$

$cost((a, c), q_1) = \frac{1000}{400} = 2.5$

$cost((a, c), q_2) = \frac{1000}{200} = 5$

$cost((a, c), q_3) = 1000$

$w = \{a, b\}$

$Q' = \{q_1, q_3\}$

$u = \{a\}$

$cost(u) = 1005$

$u = \{b\}$

$cost(u) = 1010$

$\hat{u} = \{a\}$ $(1005 < 1010)$

$perm(\{a, b\}) = (a, b)$

$cost((a, b), q_1) = \frac{1000}{200} = 5$

$cost((a, b), q_2) = \frac{1000}{250} = 4$

$cost((a, b), q_3) = 1000$

$w = \{b, d\}$

$Q' = \{q_3\}$

$u = \{b\}$

$cost(u) = 10$

$\hat{u} = \{b\}$

$perm(\{b, d\}) = (b, d)$

$cost((b, d), q_1) = 1000$

$cost((b, d), q_2) = \frac{1000}{100} = 10$

$cost((b, d), q_3) = \frac{1000}{200} = 5$

$w = \{a, b, c, d\}$

$Q' = \{q_1, q_2, q_3\}$

$u = \{a, c\}$

$cost(u) = 1007.5$

$u = \{a, b\}$

$cost(u) = 1009$

$u = \{b, d\}$

$cost(u) = 1015$

$\hat{u} = \{a, c\}$ ($1007 < 1009$ and $1007 < 1015$)

$perm(\{a, b, c, d\}) = (a, c, b, d)$

Thus $\pi_v^1 = (a, c, b, d)$

To find the next index we continue as follows:

$MCS(q_1) = \min\{1000, \frac{1000}{400}\} = 2.5$

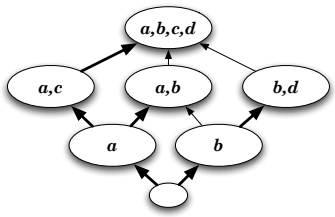
$MCS(q_2) = \min\{1000, \frac{1000}{200}\} = 5$

$MCS(q_3) = \min\{1000, \frac{1000}{1000}\} = 1000$

$r \leftarrow 2 \leq N_v = 3$

$Q_{temp} = \{q_2, q_3\}$

...



Digraph G_v for Example 7