An Integer Programming Approach for the View and Index Selection Problem

Zohreh Asgharzadeh Talebi1, Rada Chirkova2, and Yahya Fathi1

1 Operations Research Program, NC State University, Raleigh, NC 27695, {zasghar,fathi}@ncsu.edu
2 Computer Science Department, NC State University, Raleigh, NC 27695, chirkova@csc.ncsu.edu

Abstract. The view- and index-selection problem is a combinatorial optimization problem that arises in the context of on-line analytical processing (OLAP) in database management systems. After a brief introduction, we propose an integer programming (IP) model for the view- and index-selection problem and study its special structure and properties. We then use these properties to remove some variables and constraints from this IP model and obtain a model which is significantly smaller, yet its optimal solution is guaranteed to be optimal for the original problem. This allows us to solve the realistic-size instances of the problem in reasonable time using commercial IP solvers. Subsequently, we propose strategies to further reduce the size of the feasible region of this IP model although we no longer guarantee that the reduced region contains the global optimal solution. This smaller IP model can be solved much faster, but clearly its optimal solution might be suboptimal for the original problem. Finally, we carry out an extensive computational study to evaluate the effectiveness of these IP models for solving the OLAP view- and index-selection problem.

Keywords: Integer programming, Combinatorial optimization, Heuristics, Database management systems, OLAP

1 Introduction

On-line analytical processing (OLAP) and data warehousing are especially 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 or dimensions. For instance, if we consider a warehouse that keeps the information related to a company’s sales, its corresponding dimensions can be the product sold, the time of sale, the customer, the sales representative, etc.

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 Chaudhuri and Dayal (1997)).
As we shall explain later in this article, 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 the data warehouse under consideration has a star-schema structure (see Kimball and Ross (2002)) 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). We consider relational select-project-join queries with grouping and aggregation (SPJGA) (see Ullman et al. (2002)) and assume that users frequently ask a limited number of SPJGA queries for a variety of parameters (attributes), such as the itemized daily sales reports for the products, the locations, etc. Also, we assume that the time (cost) of evaluating a query is proportional to the number of stored-data rows scanned by the query-processing system when evaluating the query.

Our (original) search space of views is the view lattice defined in Harinarayan et al. (1996), which includes all star-join views with grouping and aggregation (JGA views) on the base tables. Our (original) search space of indexes includes B+-tree indexes (see Ullman et al. (2002)) on all views in the view lattice. 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 \( v \). Throughout this article we use the character \( \pi \) to represent both a fat index and the permutation vector associated with that fat index.

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. The raw-data view is a large table resulting from the join of all the stored (both fact and dimension) tables, and it contains a separate row for each transaction without any aggregation.

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 this article we introduce an integer programming (IP) model for solving this problem and propose a strategy to reduce the size of this IP model to manageable levels so that it can be solved using a commercial IP solver (such as CPLEX 10). After a brief review of related work at the end of this section, in Section 2 we propose a cost model and present a formal definition of the view- and index-selection problem. In Section 3 we present an integer programming model for this problem and study its structure. Through a series of observations and lemmas we demonstrate that we can reduce the size of the IP model by pruning a significant portion of its feasible region while keeping at least one globally optimal solution in the remaining part of the feasible region. Hence we can guarantee that an optimal solution of the resulting IP model is also optimal for the original problem. In Section 4 we offer a heuristic strategy to further reduce the size of the IP model in order to handle even larger instances of the problem, although we can no longer guarantee that the optimal solution of the resulting IP model is also optimal for the original problem. In Section 5 we present the results of an extensive computational experiment with this approach. Finally in Section 6 we offer some concluding remarks.

1.1 Related Work

The prominent role of materialized views and indexes in improving query-processing performance has long been recognized, see, for instance, Broughton (2005) and Microsoft Reference (a). Enterprise-class database-management systems that provide modules for generic view and index selection include Microsoft SQL Server (see Agrawal et al. (2000) and Microsoft Reference (b)) and DB2 (see Balmin et al. (2004), Valentin et al. (2000), and Zilio et al. (2004)). 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 it is theoretically possible. Gupta et al. (1997) show that a variant of the view- and index-selection problem is NP-hard. Furthermore, as observed by Karloff and Mihail (1999) 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. (2000) and Gupta et al. (1997).

Gupta et al. (1997) proposed two families of algorithms for solving the problem of view and index selection in a generalization of the OLAP setting. Karloff and Mihail (1999) disproved the strong performance bounds of these algorithms, by showing that the underlying approach of Harinarayan et al. (1996) cannot provide the stated worst-case performance ratios unless P=NP.

The paper by Agrawal et al. (2000) presents 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. The approach of Agrawal et al. (2000), implemented in Microsoft SQL Server, is based partly on the authors’ previous work (see Chaudhuri and Narasayya (1997)) on index selection. The contributions stated by Agrawal et al. (2000) 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.

Ezeife (1997) proposed a uniform approach for selecting views and indexes for OLAP queries. 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, e.g., Baralis et al. (1997); Kalnis et al. (2002); Shukla et al. (1998); Yang et al. (1997) and references therein, or selection of indexes only, e.g., Caprara et al. (1995); Chaudhuri et al. (2004); Kratica et al. (2003); Chaudhuri and Narasayya (1997) and references therein for OLAP. Also, Golfarelli et al. (2004) consider the problem of selecting a set of fragmented views. In their work, they propose a 0-1 integer programming model to select the best fragments which minimizes the total cost of answering the queries in the workload. Our work differs from the work in these papers in that we consider the problem of selection of views and indexes simultaneously.

2 Preliminaries

As mentioned earlier, a query $q$ can be answered using a view $v$ 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. For ease of presentation, throughout this article 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.

Example 1. 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 $q_2 \not\subseteq v$, view $v$ cannot answer query $q_2$. 
### 2.1 Cost Model

The cost model that we use is similar to the one proposed in Gupta et al. (1997), 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 $\pi$ of $v$, we only need to read the part of $v$ referenced by $\pi$ 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. 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 \textbf{GROUP BY} clause of query $q$, and $A_2$ is the set of attributes that are compared with constants in the \textbf{WHERE} clause of query $q$. Also suppose $B$ is the set of grouping attributes of view $v$. Let $\pi$ represent an index over view $v$, i.e., a permutation of attributes in the set $B$. (Recall that we use the character $\pi$ 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 $\pi$. 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 $\pi$ 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 $\pi$, i.e., the largest subset of attributes that are compared with constants in the \textbf{WHERE} clause of $q$, and form a prefix of $\pi$. Then Gupta et al. (1997) suggests the following formula to estimate the cost of answering query $q$ using view $v$ and index $\pi$

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

where \text{size}(v) represent the size of view $v$ as defined above\(^3\). For notational convenience, we use $v = \emptyset$ to represent the view which is aggregated on all of the attributes of the database, and define $\text{size}(\emptyset) = 1$. Furthermore, following the articles by Harinarayan et al. (1996), Li et al. (2005), and Gupta et al. (1997), we assume that the total cost of answering a given collection of queries is the sum of the costs of evaluating the individual queries.

\(^3\) 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 $\text{size}(v_1) \leq \text{size}(v_2)$.
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 Gupta et al. (1997). 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 paper 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 1 denotes the view whose set of grouping attributes are identical to the largest subset of \( q \) that forms a prefix of \( \pi \). (With minor modification, all results that we obtain here are valid without this assumption.)

**Example 2.** Consider a view \( v = \{a, b, c, d\} \), an index \( \pi = (a, c, b, d) \) over view \( v \), and a query \( q = \{a, b\} \). Note that according to the assumption stated above, \( a \) and \( b \) are in the WHERE clause of query \( q \) and they are compared with constants, and the GROUP BY clause of \( q \) is empty. In this example, we have \( v_\pi(q) = \{a\} \), and the estimated cost of answering query \( q \) using view \( v \) and index \( \pi \) is \( c_q(v, \pi) = \frac{\text{size}(\{a,b,c,d\})}{\text{size}(\{a\})} \).

### 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 Agrawal et al. (2000), Gupta et al. (1997)). 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 \((D, Q, b)\), where \( D \) is a database, \( Q \) is the workload (which is a set of parameterized queries), and \( b \) is the storage limit.

**Definition 1.** (Feasibility) For a problem input \((D, Q, b)\), a set of views and indexes \( \mathcal{VI} \) is feasible if (1) each query in \( Q \) 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 \((D, Q, b)\), 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 \( Q \) on the database \( Dv \), among all feasible sets of views and indexes for the problem input. Here, \( Dv \) is the database that results from adding to \( D \) the relations for all of the views and indexes in \( \mathcal{VI}^* \).
Definition 3. \((OLAP-VI\text{ problem})\) For a given problem input \((D, Q, b)\), 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 \(V^*\) (which includes the raw-data view on \(D\) and all additional views that we choose to materialize), a set \(II^*\) of indexes over the views in \(V^*\), and an association between each element of \(Q\) and its corresponding elements of \(V^*\) and \(II^*\), i.e., which view in \(V^*\) and which index in \(II^*\) (if any) should be used to answer each query in \(Q\).

Following the work of Gupta et al. (1997) and Harinarayan et al. (1996) 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.

3 An Integer Programming Model and its Properties

In this section we introduce an integer programming (IP) model for the OLAP view-and index-selection problem 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 \(IP1\)

For a given problem input \((D, Q, b)\), we define the following notation.

\[
\begin{align*}
V & : \text{ The set of all views in the view lattice} \\
\Pi(v) & : \text{ The set of all indexes of view } v, \forall v \in V \\
Q(v) & : \text{ The set of all queries in the set } Q \text{ that can be answered by view } v, \forall v \in V.
\end{align*}
\]

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 \(\pi_{jl}\), for \(l = 1, \ldots, (|v|)!\), and, for brevity, we denote the collection of all indexes associated with \(v_j\) by \(\Pi_j\). In other words, we use \(\Pi_j\) 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 \(Q\), i.e., \(Q = \{q_1, q_2, \ldots, q_m\}\). 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, \text{ and } l = 1, 2, \cdots,
\( s_{ij} = \begin{cases} 1 & \text{if view } v_j \text{ is used to answer query } q_i \text{ with no index} \\ 0 & \text{otherwise} \end{cases} \)

\( y_{ijl} = \begin{cases} 1 & \text{if view } v_j \text{ and its index } \pi_{jl} \text{ are used to answer query } q_i \\ 0 & \text{otherwise} \end{cases} \)

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 \( c_{ql}(v_j, \pi_{jl}) \), which is the estimated cost of answering query \( q_i \) using view \( v_j \) and its index \( \pi_{jl} \) 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 \( d_{ij} = \text{size}(v_j) \).

\[
\begin{align*}
\text{min} & \quad \sum_{i=1}^{m} \sum_{j \in J_i} [d_{ij}s_{ij} + \sum_{l=1}^{(|v_j|)!} c_{ijl} * y_{ijl}] \\
\text{subject to} & \quad \sum_{j \in J_i} \left[ s_{ij} + \sum_{l=1}^{(|v_j|)!} y_{ijl} \right] = 1 \quad \text{for all } i = 1 \text{ to } m \\
& \quad \sum_{j=1}^{2^k} \text{size}(v_j) \left[ t_j + \sum_{l=0}^{(|v_j|)!} x_{jl} \right] \leq b \\
& \quad x_{jl} \leq t_j \quad \text{for all } j = 1 \text{ to } 2^k, \text{ and } l = 1 \text{ to } (|v_j|)! \\
& \quad s_{ij} \leq t_j \quad \text{for all } i = 1 \text{ to } m, \text{ and } j \in J_i \\
& \quad y_{ijl} \leq x_{jl} \quad \text{for all } i = 1 \text{ to } m, \text{ and } j \in J_i, \text{ and } l = 1 \text{ to } (|v_j|)! \\
& \quad t_1 = 1 \\
& \quad t_j, s_{ij}, x_{jl}, y_{ijl} \in \{0, 1\} \quad \forall i, j, l
\end{align*}
\]

Equation (3) states that each query must be answered by exactly one view and either with no index or with exactly one of its indexes. Constraint (4) 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 (5)
states that index $\pi_{jl}$ for view $v_j$ can be materialized only if the view itself is materialized. Similarly constraint (6) states that query $q_i$ can be answered by view $v_j$ (with no index) only if this view is materialized, and constraint (7) states that query $q_i$ can be answered by view $v_j$ and its index $\pi_{jl}$ only if this index is materialized. Finally, constraint (8) simply states that the raw-data view is always selected$^4$.

### 3.2 Reducing the Size of the IP model

So far in the OLAP-VI problem and the resulting integer programming model $IP1$ we consider all views in the view lattice 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

\[ \sum_{i=1}^{m} (|J_i| + \sum_{j \in J_i} (|v_j|)!)) + \sum_{j=1}^{2^k} (|v_j|)! + 2^k \text{ variables and } \sum_{i=1}^{m} (|J_i| + \sum_{j \in J_i} (|v_j|)!)) + \sum_{j=1}^{2^k} (|v_j|)! + m + 2 \text{ constraints in the integer programming model } IP1. \]

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 solvers such as CPLEX 10. 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 this section 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.

### 3.2.1 Reducing the set of views

We start by making a few observations regarding the properties of the views that appear in an optimal solution for a given OLAP-VI problem. Proofs of these observations and the lemma are straightforward and, for brevity, we do not include them in this article. For detail discussion see the work by Asgharzadeh Talebi (2009).

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

**Observation 2.** Given an instance of the OLAP-VI problem with input data $(D, Q, b)$, 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 $v$.

$^4$ In our IP model we denote this raw-data view by $v_1$.  

It follows that we can easily reduce the search space of views in the \textit{OLAP-VI} problem by removing from the set \( V \) every view \( v \) that satisfies the condition of either observation\(^5\), and the resulting problem (with the smaller collection of views in its search space) is guaranteed to have at least one optimal solution which is also optimal for the original \textit{OLAP-VI} problem. We refer to this (reduced) set of views as \( V' \).

Obviously the task of determining the set \( V' \) itself requires some effort. In order to evaluate the corresponding computational requirements 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 \( 1 \leq r \leq |Q| \)) 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 \( O(2^{|Q|}) \), while the computational requirement of the second method is of order \( O(|Q|2^k) \). Depending on the specific values of \( k \) and \( |Q| \), we choose either the first or the second method, whichever results in smaller computational effort.

### 3.2.2 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 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 \( w_1 \subseteq w_2 \) 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.

**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 \( \pi \) is associated with path \( P \) if the set of attributes of each and every node in \( P \) is a prefix of \( \pi \).

**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 \( w_v \). Suppose the order of the nodes on path \( P \)

\(^5\) Note that in model IP1 we have already excluded the views covered by the first observation through defining the subsets \( V_i \) and \( J_i \).
is \( \emptyset, w_1, w_2, \ldots, w_{i-1}, w_i, w_{i+1}, \ldots, w_s \). 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 1.** Given a view \( v \) and its corresponding digraph \( G_v \), if two indexes \( \pi_1 \) and \( \pi_2 \) 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 \( \pi_1 \) and \( \pi_2 \) 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 \( \pi_1 \) is the same as the largest subset of \( q \) that forms a prefix of \( \pi_2 \), i.e., \( v_{\pi_1}(q) = v_{\pi_2}(q) \). As a result, \( c_q(v, \pi_1) = c_q(v, \pi_2) \). \( \square \)

**Lemma 2.** Given a view \( v \) and its corresponding digraph \( G_v \), if an index \( \pi \) of view \( v \) is not associated with any source-sink path in digraph \( G_v \), then there exists another index \( \pi' \) 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 \( \pi \). Let \( w \) be the last node in \( P(\pi) \) and let \( r = |w| \). Since \( \pi \) 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 \( \pi \) 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 \( w_{r+1}, w_{r+2}, \ldots, w_{r+t} \).

Suppose \( \pi' \) is an index associated with \( P \). Also, suppose the order of the first \( r \) attributes in \( \pi' \) is the same as the order of the first \( r \) attributes in \( \pi \). 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 \( \pi \) and \( \pi' \) have all of the attributes of \( w \) as their 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 \( \pi \) and \( \pi' \) 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 \( \pi' \) is associated with \( P \), all of the attributes of \( w_h \) form a prefix of \( \pi' \). 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 \( q \in Q(v) \).  

![Fig. 1. Digraph \( G_v \) for Example 3.](image)

From Lemma 1 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 2 it follows that for each index \( \pi \) of view \( v \) that is not associated with any source-sink path of digraph \( G_v \), we can find an index \( \pi' \) of view \( v \) which is associated with a source-sink path of \( G_v \) and is at least as effective as \( \pi \) 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 \( \pi_i \). We define \( \Pi'(v) \) as the collection of all indexes for view \( v \) obtained in this manner.

Following is a small illustrative example.

**Example 3.** 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 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 forth 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 \( q \in Q \), 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 \( q \) with this view. It follows that we can easily 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 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 a few randomly constructed instances at the end of this section. 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. For each view \( v \), the computational requirement for constructing the corresponding digraph \( G_v \) is of order \( O(\min\{s^{v\mid v}, s^{Q(v)}\}) \).

### 3.3 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:

\[
\min \sum_{i=1}^{m} \sum_{j \in J'_i} \left[ d_{ij} s_{ij} + \sum_{l \in L'_j} c_{ijl} y_{ijl} \right] \quad IP2
\]
subject to \[
\sum_{j \in J'} \left[ s_{ij} + \sum_{l \in L'_j} y_{ijl} \right] = 1 \quad \text{for all } i = 1 \text{ to } m
\]
\[
\sum_{j \in J'} \text{size}(v_j) \left[ t_j + \sum_{l \in L'_j} x_{jl} \right] \leq b
\]
\[
x_{jl} \leq t_j \quad \text{for all } j \in J' \text{ and } l \in L'_j
\]
\[
s_{ij} \leq t_j \quad \text{for all } i = 1 \text{ to } m \text{ and } j \in J'_i
\]
\[
y_{ijl} \leq x_{jl} \quad \text{for all } i = 1 \text{ to } m, j \in J'_i, \text{ and } l \in L'_j
\]
\[
t_1 = 1
\]
\[
t_j, s_{ij}, x_{jl}, y_{ijl} \in \{0, 1\} \quad \forall i, j, l
\]

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

**Theorem 1.** Given an OLAP-VI problem with input data \((D, Q, b)\), 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 \(IP2\).

### 3.4 Numeric Examples

In this section we compare the sizes of models \(IP1\) and \(IP2\) for a few 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. The databases that we used in our instances are a 7-attribute and a 13-attribute TPC-H databases (see TPC-H Revision 2.1.0 (2002)). We shall discuss these databases and the procedure that we used to build the collection of instances later in Section 5.

Our first collection consists of ten instances over the 7-attribute database and the results are presented in Table 1. For each instance (each row) we report the number of queries in that instance\(^6\) and the corresponding number of views and indexes in each of the models \(IP1\) and \(IP2\), respectively. Table 2 contains similar results for a collection of ten instances

---

\(^6\) As we will explain in Section 5, queries are generated randomly.
over the 13-attribute database. We observe that the number of views and indexes in model $IP2$ is significantly smaller than those in model $IP1$, and the difference becomes larger as the size of the problem gets larger.

<table>
<thead>
<tr>
<th>instance</th>
<th>number of queries</th>
<th>number of views</th>
<th>number of indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$IP1$</td>
<td>$IP2$</td>
<td>$IP1$</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>128</td>
<td>45</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>128</td>
<td>45</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>128</td>
<td>56</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>128</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>128</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>128</td>
<td>40</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>128</td>
<td>42</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>9</td>
<td>40</td>
<td>128</td>
<td>58</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>128</td>
<td>64</td>
</tr>
</tbody>
</table>

Table 1. Comparison of the number of views and indexes in models $IP1$ and $IP2$ for the instances over a 7-attribute TPC-H database.

<table>
<thead>
<tr>
<th>instance</th>
<th>number of queries</th>
<th>number of views</th>
<th>number of indexes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$IP1$</td>
<td>$IP2$</td>
<td>$IP1$</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>8,192</td>
<td>27</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>8,192</td>
<td>28</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>8,192</td>
<td>25</td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>8,192</td>
<td>29</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>8,192</td>
<td>49</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>8,192</td>
<td>60</td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td>8,192</td>
<td>165</td>
</tr>
<tr>
<td>18</td>
<td>30</td>
<td>8,192</td>
<td>305</td>
</tr>
<tr>
<td>19</td>
<td>40</td>
<td>8,192</td>
<td>556</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>8,192</td>
<td>725</td>
</tr>
</tbody>
</table>

Table 2. Comparison of the number of views and indexes in models $IP1$ and $IP2$ for the instances over a 13-attribute TPC-H database.

## 4 An IP-based Heuristic Procedure

In this section we propose a strategy to further reduce the size of the integer programming model for the view- and index-selection problem. In this context we limit the choice of indexes for each view $v \in V'$ (Section 3.2) to a relatively small collection of promising alternatives for this view, which we refer to as $\Pi''(v)$. In our experiment this collection was consistently significantly smaller than the corresponding collection $\Pi'(v)$ that we defined in Section 3.2. This, in turn, would allow us to solve larger instances of the OLAP-VI problem. We refer to this smaller integer programming model as $IPN$. The downside of this approach,
However, is that we can no longer guarantee that an optimal solution of the resulting integer programming model (IPN) is an optimal solution of the original OLAP-VI problem.

More specifically, for each view \( v \in V' \) we limit the number of indexes associated with this view to a relatively small positive integer that we refer to as \( N_v \). We define \( N_v = |Q(v)| \). 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 3.2 (note that each query \( q \in Q(v) \) can be answered optimally with respect to \( v \) by at most one index of \( v \)).

We select the \( N_v \) indexes associated with view \( v \) in a greedy manner as follows:

\[
\begin{align*}
\pi_1^v &= \arg\min_{\pi \in \Pi'(v)} \sum_{q \in Q(v)} c_q(v, \pi) \\
\pi_2^v &= \arg\min_{\pi \in \Pi'(v)} \sum_{q \in Q(v)} \min\{c_q(v, \pi), c_q(v, \pi_1^v)\} \\
\pi_3^v &= \arg\min_{\pi \in \Pi'(v)} \sum_{q \in Q(v)} \min\{c_q(v, \pi), c_q(v, \pi_1^v), c_q(v, \pi_2^v)\} \\
&\vdots \\
\pi_{N_v}^v &= \arg\min_{\pi \in \Pi'(v)} \sum_{q \in Q(v)} \min\{c_q(v, \pi), c_q(v, \pi_1^v), c_q(v, \pi_2^v), \ldots, c_q(v, \pi_{N_v-1}^v)\}
\end{align*}
\]

and define the set \( \Pi''(v) = \{\pi_1^v, \pi_2^v, \ldots, \pi_{N_v}^v\} \). (If the total number of indexes for view \( v \), i.e. \( |\Pi'(v)| \), is less than \( |Q(v)| \), we select all indexes in \( \Pi'(v) \) for this view.)

**Model IPN**

We define the 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 to determine the set \( \Pi''(v) \) for each view \( v \in V' \); we refer to this algorithm as Algorithm IPNIDX.

**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 \( \pi_1^v \), in the second iteration we find \( \pi_2^v \), 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 \( w \) (\( \text{perm}(w) \)), based on the order of the attributes of one of its parent nodes. We refer to this order as \( \text{perm}(w) \). The last node considered in this topological order is the sink node \( v \), and we declare the corresponding order \( \text{perm}(v) \) as the index selected in this iteration.

At the end of each iteration, we update the value of \( \text{MCS}(q) \) for each \( q \in Q(v) \), where \( \text{MCS}(q) \) is the minimum cost of answering query \( q \) using the indexes selected so far. At the beginning of the first iteration, \( \text{MCS}(q) = \text{size}(v) \) for all queries in \( Q(v) \).
Algorithm 1: IPNIDX

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

\textbf{Input} : view \( v \), digraph \( G_v \), and the set of queries \( Q(v) \) \hspace{1cm} \( \setminus \) See Section 3.2 for construction of \( G_v \).
\textbf{Output}: all elements of the set \( \Pi'(v) \)

\begin{algorithm}
\begin{algorithmic}
\State \( MCS(q) = \text{size}(v) \) for all \( q \in Q(v) \)
\State \( cost(\emptyset, q) = \text{size}(v) \) for all \( q \in Q(v) \)
\State \( Q_{\text{temp}} \leftarrow Q(v) \)
\State \( G_r \leftarrow G_v \)
\State \( r \leftarrow 1 \)
\While {\( r \leq N_v \)}
\For {each node \( w \) (other than \( \emptyset \)) in \( G_v \) in topological order}
\State \( Q' \leftarrow \{ q \in Q_{\text{temp}} | q \cap w \neq \emptyset \text{ and } q \cap w \neq w \} \)
\For {each node \( u \) which is a parent node of \( w \)}
\State \( \hat{u} = \arg \min_{u \in \text{parent}(w)} \text{cost}(u) \)
\State \( \text{perm}(w) = \text{perm}(\hat{u}) + \text{arb}(w \setminus \hat{u}) \)
\State \( \text{cost}(w, q) = c_q(v, \text{perm}(w)) \) for each \( q \in Q_{\text{temp}} \)
\EndFor
\State \( \text{return} \text{perm}(v) \)
\State \( MCS(q) = \min\{\text{cost}(v, q), MCS(q)\} \) for each \( q \in Q_{\text{temp}} \)
\State \( Q_{\text{temp}} \leftarrow \{ q \in Q_{\text{temp}} | \text{perm}(v)(q) \neq q \} \).
\State \( r \leftarrow r + 1 \)
\EndFor
\EndWhile
\end{algorithmic}
\end{algorithm}

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

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 \( Q' = \{ q \in Q_{\text{temp}} | q \cap w \neq \emptyset \text{ and } q \cap w \neq w \} \). Note that query \( q \in Q(v) \) is in \( Q_{\text{temp}} \) 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 \( \text{perm}(w) = (\text{perm}(u), \text{arb}(w \setminus u)) \), where \( u \) is one of the parent nodes of node \( w \), and \( \text{arb}(w \setminus u) \) 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 \( \text{perm}(w) \) we have \( \text{cost}(u, q) \) for each parent node \( u \) and for each query in \( Q' \); thus we can compute \( \text{cost}(u) = \sum_{q \in Q(v)} \min \{ \text{cost}(u, q), MCS(q) \} \) for each parent node \( u \) of node \( w \).

Given the digraph \( G_v \) for view \( v \), the computational requirement of Algorithm IPNIDX is \( O(N_v \times |Q(v)| \times 4^\min(|Q(v)|, |v|)) \).

4.1 Numeric Examples

In this section we compare the number of indexes in models \( IP2 \) and \( IPN \) for the same collection of instances that we mentioned in Section 3.4. We report the results for the instances over the 7-attribute database and the 13-attribute database in Table 3 and Table 4, respectively.

\footnote{To 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. Thus, the computational requirement of Algorithm IPNIDX is \( O(N_v \times |Q(v)| \times 4^\min(|Q(v)|, |v|)) \).}
<table>
<thead>
<tr>
<th>instance</th>
<th>number of queries</th>
<th>number of indexes</th>
<th>IP2</th>
<th>IPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>913</td>
<td>150</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>500</td>
<td>132</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>561</td>
<td>172</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>521</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>713</td>
<td>161</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>694</td>
<td>113</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>418</td>
<td>125</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td>802</td>
<td>259</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>40</td>
<td>1,356</td>
<td>324</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>2,117</td>
<td>408</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Comparison of the number of indexes in models IP2 and IPN for the instances over a 7-attribute TPC-H database.

<table>
<thead>
<tr>
<th>instance</th>
<th>number of queries</th>
<th>number of indexes</th>
<th>IP2</th>
<th>IPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>10</td>
<td>1,218</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>409</td>
<td>77</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>534</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>358</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>5,437</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>7,069</td>
<td>203</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td>15,460</td>
<td>619</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>30</td>
<td>3,061,155</td>
<td>1,207</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>40</td>
<td>3,177,400,892</td>
<td>2,464</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>3,166,128,428</td>
<td>4,345</td>
<td></td>
</tr>
</tbody>
</table>

Table 4. Comparison of the number of indexes in models IP2 and IPN for the instances over a 13-attribute TPC-H database.
We observe that for each instance the number of indexes in model $IPN$ is significantly smaller than the corresponding number in model $IP2$, especially for larger instances. In the next section we study the effect of this difference on the corresponding execution times required to solve the models using a standard IP solver, and on the quality of the solutions obtained.

5 Experimental Results

We now present the results of a computational experiment with the models presented in this article. Our objectives in this experiment are as follows:

1- Evaluate the scalability of the exact model $IP2$.
2- Evaluate the scalability of the inexact model $IPN$ and the quality of the solutions obtained by this model.
3- Compare the effectiveness of the inexact model $IPN$ with an existing heuristic procedure proposed by Gupta et al. (1997).

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 TPC-H Revision 2.1.0 (2002)). We then solve each instance using different models and procedures as applicable and report our findings.

We developed all programs in C++ and ran them on a PC with a 3GHz Intel P4 processor, 1GB RAM, cache size of 512 KB, and a 80GB hard drive running Red Hat Linux Enterprise 4. We used CPLEX 10 by ILOG (2004) to solve the integer programming models.

5.1 Data sets

Each instance of the OLAP-VI problem is identified by a given database $D$, a given collection of queries $Q$, and a given storage space $b$. We used two different databases of the TPC-H benchmark (see TPC-H Revision 2.1.0 (2002)). 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 make up 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. Agrawal et al. (2000); Chaudhuri et al. (2004); Chaudhuri and Narasayya (1997); Kalnis et al. (2002)).

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.\(^8\)

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 view plus the size of the queries in the set \( Q \). Suppose the value of storage space \( b \) is expressed as:

\[
b = \text{size}(v_1) + \alpha \times \left( \sum_{q \in Q} \text{size}(q) \right)
\]

(10)

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 \( Q \) 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 \( \alpha = 0.1, 0.2, 0.3, \ldots, 0.9, 1 \), the pattern of findings did not change very much, although the actual solution did change as expected.

### 5.2 Results

Our first experiment consists of solving a collection of instances of the OLAP-VI problem using the exact model \( IP2 \) and the inexact model \( IPN \). We solved ten instances over the 7-attribute database and ten instances over the 13-attribute database with the number of queries ranging between 10 and 50.\(^9\) We report our findings in Tables 5 and 6, respectively. For each instance we report (1) the number of queries, (2) the optimal value of the corresponding integer programming models \( IP2 \) and \( IPN \) (all integer programming models are solved using CPLEX 10), (3) the execution time, which includes both the pre-processing time to construct the restricted view sets and index sets in the IP models as well as the solving time using CPLEX 10. For these collections of instances, we make the following observations:

- The execution time for solving model \( IP2 \) is relatively small for small to moderate size instances, but for larger instances, the execution time increases rapidly. For the largest instance in Table 5 (instance 10), the execution time is over eight minutes, and for the large instances in Table 6 (instances 17 through 20) the execution time exceeds our one hour time limit. Obviously, for all instances where the execution was completed, the reported cost for model \( IP2 \) is optimal for the corresponding OLAP-VI problem.

\(^8\) Consistent with the assumption that we made earlier, we continue to assume that for each query all associated attributes are in its \texttt{WHERE} clause and they are compared with constants.

\(^9\) These instances are the exact same instances that we used in Sections 3.4 and 4.1.
<table>
<thead>
<tr>
<th>instance</th>
<th>number of queries</th>
<th>optimal cost</th>
<th>execution time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IP2</td>
<td>IPN</td>
<td>IP2</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>20.25</td>
<td>20.25</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>20.41</td>
<td>22.17</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>20.29</td>
<td>20.54</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>20.42</td>
<td>20.66</td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>20.41</td>
<td>20.52</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>20.17</td>
<td>21.00</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>21.05</td>
<td>21.16</td>
</tr>
<tr>
<td>8</td>
<td>30</td>
<td>30.29</td>
<td>30.30</td>
</tr>
<tr>
<td>9</td>
<td>40</td>
<td>40.35</td>
<td>50.57</td>
</tr>
<tr>
<td>10</td>
<td>50</td>
<td>50.16</td>
<td>50.30</td>
</tr>
</tbody>
</table>

Table 5. Comparison of (1) the costs obtained from solving models IP2 and IPN, and (2) the total execution times for solving each of the models IP2 and IPN (including the times required to construct the set of views and indexes) for each of the instances over the 7-attribute TPC-H database.

<table>
<thead>
<tr>
<th>instance</th>
<th>number of queries</th>
<th>optimal cost</th>
<th>execution time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IP2</td>
<td>IPN</td>
<td>IP2</td>
</tr>
<tr>
<td>11</td>
<td>10</td>
<td>10.00</td>
<td>10.00</td>
</tr>
<tr>
<td>12</td>
<td>10</td>
<td>10.07</td>
<td>10.07</td>
</tr>
<tr>
<td>13</td>
<td>10</td>
<td>10.00</td>
<td>10.00</td>
</tr>
<tr>
<td>14</td>
<td>10</td>
<td>10.29</td>
<td>10.64</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>15.11</td>
<td>15.44</td>
</tr>
<tr>
<td>16</td>
<td>15</td>
<td>15.12</td>
<td>15.23</td>
</tr>
<tr>
<td>17</td>
<td>20</td>
<td>-</td>
<td>20.63</td>
</tr>
<tr>
<td>18</td>
<td>30</td>
<td>-</td>
<td>30.11</td>
</tr>
<tr>
<td>19</td>
<td>40</td>
<td>-</td>
<td>40.14</td>
</tr>
<tr>
<td>20</td>
<td>50</td>
<td>-</td>
<td>50.01</td>
</tr>
</tbody>
</table>

Table 6. Comparison of (1) the costs obtained from solving models IP2 and IPN, and (2) the total execution times for solving each of the models IP2 and IPN (including the times required to construct the set of views and indexes) for each of the instances over the 13-attribute TPC-H database.
The execution time for solving the inexact model IPN is significantly smaller than that of IP2. Also, the cost of the solution obtained via model IPN is close to the optimal cost for those instances where we know the optimal solution (obtained via the exact model IP2). For this collection of instances, the average cost obtained via model IPN is 1% more than the corresponding optimal cost, and the maximum deviation from the optimal cost is 9% (instance 2 in Table 5). For the remaining instances where we do not know the optimal cost (instances 17 through 20 in Table 6), we observe that the cost obtained via IPN is at most 4% larger than a corresponding lower bound (i.e., the number of queries).10

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 (about 45 minutes), and the value of cost obtained from this model is 100.06. Needless to say that we could not solve model IP2 for this instance since its execution time would have been excessive.

In all instances we also observed that the pre-processing time used to build each of the models IP2 and IPN is significantly smaller than the corresponding CPLEX time used to solve that model. For brevity, the exact times are not reported in Tables 5 and 6.

In our second experiment we solved a collection of instances of the OLAP-VI problem using a well-known heuristic procedure proposed by Gupta et al. (1997), and compared the results with those obtained using our inexact model IPN.11 The approach of Gupta et al. (1997) is a greedy algorithm that we refer to as algorithm GHRU.12

This collection consists of six instances over the 7-attribute database, and each instance has ten queries. The results are displayed in Table 7. 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:

<table>
<thead>
<tr>
<th>instance</th>
<th>IPN cost</th>
<th>GHRU cost</th>
<th>IPN execution time (sec.)</th>
<th>GHRU execution time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>12.52</td>
<td>1,173,633.70</td>
<td>0.93</td>
<td>817.85</td>
</tr>
<tr>
<td>22</td>
<td>53.96</td>
<td>1,199,286.60</td>
<td>0.61</td>
<td>524.49</td>
</tr>
<tr>
<td>23</td>
<td>31.59</td>
<td>900,728.16</td>
<td>0.63</td>
<td>615.02</td>
</tr>
<tr>
<td>24</td>
<td>11.68</td>
<td>1,199,262.84</td>
<td>0.60</td>
<td>641.09</td>
</tr>
<tr>
<td>25</td>
<td>12.15</td>
<td>1,473,421.00</td>
<td>0.60</td>
<td>571.19</td>
</tr>
<tr>
<td>26</td>
<td>14.85</td>
<td>1,199,290.98</td>
<td>0.60</td>
<td>523.90</td>
</tr>
</tbody>
</table>

Table 7. 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.

10 Note that since all attributes of each query are in its WHERE clause and they are compared with constants (by assumption), it follows that the corresponding cost of answering this query using an appropriate view and a proper index could be as low as 1 (i.e., only one row of the corresponding view needs to be scanned). Hence, the total cost of answering a given collection of queries could be as low as the number of queries.

11 As explained in Section 2.1, we employed the cost model proposed by Gupta et al. (1997), hence our results are directly comparable.

12 From the two algorithms proposed by Gupta et al. (1997), we choose the r-greedy algorithm which has a better performance guarantee, and we set r = 4 for the parameter of this algorithm as suggested in Gupta et al. (1997).
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.

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 $IP2$ 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 Conclusion

In this article we undertook a systematic study of the OLAP view- and index-selection problem through an integer programming model for the problem, and proposed several algorithms to prune the space of potentially beneficial views and indexes. Our specific contributions are as follows:

- We introduced an integer programming model for the OLAP-VI problem.
- We developed an algorithm that effectively and efficiently prunes the space of potentially beneficial views and indexes while keeping at least one globally optimal solution in the
search space. Thus the modified integer-programming model is guaranteed to find an optimal solution.

- We developed a heuristic procedure 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 the global optimality of the resulting solution.
- We presented an experimental evaluation of our algorithms and compared our inexact approach with the well-known greedy approach of Gupta et al. (1997).
- These experiments show that our proposed integer programming approach results in high-quality solutions — in fact, in globally optimal solutions for many realistic-size problem instances. Thus, it compares favorably with the well-known OLAP-centered approach of Gupta et al. (1997).

Our contributions open new avenues for view and index selection and materialization in OLAP and other systems. Specifically, this project lays the foundation for studying view and index selection 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 it has become clear that advances in view or index selection and advances in query rewriting using views and indexes are interrelated, which implies that these problems need to be studied together (see the work by Afrati and Chirkova (2005)).

References


