Exact and Inexact Methods for Selecting Views and Indexes for OLAP Performance Improvement

Zohreh Asgharzadeh Talebi
Operations Research Program
NC State University
Raleigh, NC 27695 USA
zasghar@ncsu.edu

Rada Chirkova
Computer Science Dept.
NC State University
Raleigh, NC 27695 USA
chirkova@csc.ncsu.edu

Yahya Fathi
Operations Research Program
NC State University
Raleigh, NC 27695 USA
fathi@ncsu.edu

Matthias Stallmann
Computer Science Dept.
NC State University
Raleigh, NC 27695 USA
matt_stallmann@ncsu.edu

ABSTRACT

In on-line analytical processing (OLAP), precomputing (*materializing as views*) and *indexing* auxiliary data aggregations is a common way of reducing query-evaluation time costs for important data-analysis queries. We consider an OLAP view- and index-selection problem stated as an optimization problem, where (i) *the inputs* include 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 costs of the input queries, subject to the storage limit. While greedy and other heuristic strategies for choosing views or indexes might have some success in improving the costs, it is highly nontrivial to arrive at a globally optimum solution, one that reduces the processing costs of typical OLAP queries as much as is theoretically possible. In fact, as observed in [16] and to the best of our knowledge, there is no known approximation algorithm for OLAP view or index selection with nontrivial performance guarantees.

In this paper we propose a systematic study of the OLAP view- and index-selection problem. Our specific contributions are as follows: (1) we develop an algorithm that effectively and efficiently prunes the space of potentially beneficial views and indexes given realistic-size instances of the problem; (2) we provide formal proofs that our pruning algorithm keeps at least one globally optimum solution in the search space, thus the resulting integer-programming model is guaranteed to find an optimal solution; (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 the global optimality of the resulting solution; and (4) we present an experimental comparison of our proposed approaches with the state-of-the-art approaches of [2, 11]. Our experiments show that our proposed approaches to view and index selection result in high-quality solutions — in fact, in *globally optimum* solutions for many realistic-size problem instances. Thus, they compare favorably with the well-known OLAP-centered approach of [11] and provide for a winning combination with the end-to-end framework of [2] for generic view and index selection.

1. INTRODUCTION

On-line analytical processing (OLAP) and data warehousing are essential elements of decision support, which is aimed at enabling executives, managers, and analysts to make better and faster decisions [8]. OLAP applications include marketing, business reporting for sales, management reporting, business-process management, budgeting and forecasting, financial reporting, and health care. Stored OLAP data are commonly presented as a large-scale multidimensional *data cube;* typical data-analysis queries on the cube data involve aggregation of large volumes of stored data and are thus complex and time consuming. Precomputing (*materializing as views*) and *indexing* auxiliary data aggregations is a common way in OLAP of reducing query-evaluation time costs for frequent and important data-analysis queries. We now give an example that shows how materialized views and indexes may speed up the evaluation times of aggregate queries. [1]

EXAMPLE 1.1. *Consider a data warehouse with three stored relations:* `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.*

[1]Throughout the paper we assume that data warehouses under consideration have the star schema [8], and that the costs of evaluating a query are proportional to the number of stored-data tuples scanned by the query-processing system when evaluating the query [11, 13].

Suppose the query workload has two queries, Q1 *and* Q2. Q1 *asks for the maximal quantity of products sold per state in November 2006.* Q2 *asks for the total quantity of products sold per company per year in North Carolina since 2000. The SQL representation of the queries is as follows:*

```
Q1: SELECT State, MAX(QtySold)
    FROM Sales s, Time t, Company c
    WHERE s.DateID = t.DateID AND s.CID = c.CID
    AND Year = 2006  AND Month = 'Nov'
    GROUP BY State;

Q2: SELECT c.CID, Year, SUM(QtySold)
    FROM Sales s, Time t, Company c
    WHERE s.DateID = t.DateID AND s.CID = c.CID
    AND State = 'NC' AND Year > 2000
    GROUP BY c.CID, Year;
```

Each of Q1 *and* Q2 *can be answered using either the original stored fact and dimension tables or the raw-data view*[2] *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 time costs; the same point holds about* Q2.*)*

We can use techniques from [1, 13] to show that the following view V *can be used to give exact answers to each of* Q1 *and* Q2.

```
V: SELECT c.CID AS CID, Year, Month, State,
   SUM(QtySold) AS SumQS, MAX(QtySold) AS MaxQS
   FROM Sales s, Time t, Customer c
   WHERE s.DateID = t.DateID AND s.CID = c.CID
   GROUP BY c.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 using* 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* Year, Month, State, CID *of the table* V, *we may be able to drastically cut the evaluation costs for* Q1. *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 2006.) At the same time, in answering query* Q2 *on view* V, *index* I *may not be as beneficial, because* Year *is the only attribute that can be used to limit the number of tuples scanned for* Q2. □

The prominent role of materialized views and indexes in improving query-processing performance has long been recognized, see, for instance, [5, 20]. Enterprise-class database-management systems that include modules for *generic* view and index selection include Microsoft SQL Server [2, 21] and DB2 [5]. 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 optimum solution,* one that reduces the processing costs of typical OLAP queries as much as is theoretically possible.

As we show in our experiments (Section 7) as well as in our related project [18], our proposed approaches to view and index selection result in high-quality solutions — in fact, in *globally optimum* solutions for many realistic-size problem instances. Thus, they compare favorably with the well-known OLAP-centered approach of [11] and provide for a winning combination with the end-to-end framework of [2] for generic view and index selection.

Of course, we can talk about "globally optimum" solutions only when we have defined the OLAP view- and index-selection problem in a formal way. We provide a precise definition of the OLAP view- and index-selection and a detailed discussion in Section 2; we now give an informal discussion of this optimization problem.

In our OLAP view- and index-selection problem, (1) *the inputs* include the data-warehouse schema, a set of data-analysis queries of interest, and a storage-limit constraint, and (2) *the output* is a set of definitions of those views and indexes that minimize the cost measure for the input queries, subject to the input constraint. In our work, we do not consider the maintenance cost; we only consider the view selection problem under the storage space limitation. For each instance of this optimization problem one can find a globally optimum solution, for instance by complete enumeration of all potential solutions. Typically (see [11]), the search space of views and indexes that can be in a potential solution includes (1) views defined by the star join followed by a `GROUP BY` on some dimension attributes in the input warehouse schema, and (2) B+-tree indexes on such views, where each index is defined as a sequence of `GROUP BY` attributes in the view definition. For realistic-size problem instances the size of this search space tends to be very large, thus finding optimum solutions w.r.t. the entire search space is infeasible by brute-force methods. In fact, NP-completeness of a variant of the problem described here is proved in [11]. Thus, it is natural to look for heuristic solutions. Well-known past efforts in this direction include [2, 11]; we discuss these approaches in detail in Section 1.1.

In contrast with the past heuristic approaches, we propose a systematic study of the OLAP view- and index-selection problem. As observed in [16] and to the best of our knowledge, there is no known approximation algorithm for view or index selection (given a search space of views and indexes) with nontrivial performance guarantees on OLAP data cubes. This is one reason that in this paper we concentrate instead on the problem of *pruning* the search space of views and indexes, with the hope that the resulting search space is small enough, for practical problem instances, that we can use software such as CPLEX [14] to get optimum or

[2]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.

near-optimum solutions with respect to that resulting search space. (As we show in this paper, some of our pruning methods keep in the search space at least one globally optimum solution, which can then be found by CPLEX.)

Our specific contributions:

- We develop an algorithm that effectively and efficiently prunes the search space of potentially beneficial views and indexes, for realistic-size instances of the problem (Section 4). The pruned search space significantly reduces the size of our integer-programming (IP) model, so that it can be solved more efficiently by an integer program solver such as CPLEX [14].

- 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 via CPLEX (or any other IP solver) is guaranteed to be globally optimal. This includes many problem instances of practical interest.

- 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 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 and CPLEX is optimal for the original problem. Thus, our proposed algorithms are IP-based inexact methods (heuristic procedures) for solving the OLAP view- and index-selection problem.

- We present an experimental comparison of our IP-based inexact approach with the generic state-of-the-art view- and index-pruning approach of [2]. Our experiments show that for the special case of OLAP queries, the model size of our approach is small enough to make solution by CPLEX tractable while still retaining solutions that have significantly better cost than those produced when [2] is used as a heuristic to reduce the search space. Furthermore, the runtime required to do our reduction is not significantly larger than that of [2].

- Further, we report the results of some experiments where we compare the performance of the well-known view- and index-selection approach of [11] with the performance of all IP-based approaches that we propose here. Of course, if the size of the problem instance permits us to use the IP model with the original (or pruned) search space, then it is always preferable to do so (as opposed to using the heuristic approach of [11]), since it guarantees to obtain a globally optimal solution. However, in the instances where the size of this original (or pruned) search space is so large that we cannot use the exact IP model, then in the IP-based approach we also employ a reduced search space, hence we can no longer guarantee global optimality. In this case, a direct comparison of the IP-based heuristic with that of [11] is not possible. In section 7 we report the results of applying our IP-based heuristic and the approach of [11] on a collection of several instances.

- In developing our solutions, we take advantage of the special structure of data cubes.

- As our experiments show that we can solve realistic-size instances of the problem in a fairly short amount of time using our approaches, it is easy to implement our approaches into standard database systems.

The remainder of this paper is organized as follows. We review related work in Section 1.1. In Section 2 we discuss the formulation and settings for our OLAP view- and index-selection optimization problem, and give a high-level overview of our proposed approaches. Section 3 presents our integer-programming model for view and index selection given a (possibly pruned) search space of views and indexes. In Section 4 we propose approaches to prune the 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 sections 5 and 6 we propose methods for further reducing the size of the search space in order to reduce the size of the resulting IP model, but we no longer guarantee global optimality. In Section 7 we present and discuss our experimental results. We conclude in Section 8.

1.1 Related Work

Recall (see, e.g., [1]) that in selecting views or indexes that would improve query-processing performance, producing solutions that would guarantee user-specified quality (in particular, globally optimum solutions) with respect to all potentially beneficial indexes and views is a computationally hard problem. In general, the authors of the past approaches have concentrated on experimental demonstrations of the quality of their solutions. A notable exception is the line of work including [11, 12, 13]. In particular, a well-known paper [11] by Gupta and colleagues proposed two families of algorithms for solving the problem of view and index selection in a generalization of the OLAP setting. Unfortunately, in 1999 the paper [16] disproved the strong performance bounds of these algorithms, by showing that the underlying approach of [13] cannot provide the stated worst-case performance ratios unless P=NP. As observed in [16] and to the best of our knowledge, there is no known approximation algorithm for view or index selection with nontrivial performance guarantees on data cubes.

[10] discusses 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. The paper [10] leaves out most implementation detail as well as any performance study of the proposed approach, which makes the approach very hard to compare with other work.

The state-of-the-art paper [2] presents a 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 [2], implemented in Microsoft SQL Server, is based partly on the authors' previous work [9] on index selection. The contributions stated in [2] are (i) the proposed end-to-end framework for view and index selection in practical systems, and (ii) the module for building the search space of potential views and indexes for a given query workload. (Interestingly, the authors of [2] do not recognize as a contribution their heuristic algorithm for selecting views and indexes from the search space built in their framework.) In this paper, 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 of [2]. This means that our algorithms are suitable for complementing the overall framework of [2] in the special case of OLAP, by providing the user with solution-quality guarantees on the views and indexes to be materialized. Also see [18] for our approach to quality-guaranteed view and index selection for the [2] framework, for the general case of typical practical (both OLAP and OLTP) query, view, and index classes.

Other past work considers either selection of indexes only (see, e.g., [6, 7] and references therein) or selection of views only (see, e.g., [4, 15, 22, 24] and references therein) for OLAP. In particular, Yang and colleagues [24] propose an integer-programming model for selecting the search space of views, coupled with a heuristic algorithm for selecting views from the resulting space, for the cost measure of query-processing costs combined with view-maintenance costs. Note that in our approach we use integer programming at the stage of view-selection proper, rather than at the stage of forming or pruning the search space, and that our search space includes not only views but also indexes.

2. PRELIMINARIES

We consider relational select-project-join queries with grouping and aggregation (SPJGA) in star-schema data warehouses [8, 17]. Similarly to [11, 13, 15, 22], 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 (full) search space of views is the *view lattice* defined in [13], 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 used in the `GROUP BY` and `WHERE` clauses in the input queries, and (2) has aggregation on *all* the attributes aggregated in the input queries, using all the aggregation functions used in the queries (such views are called "multiaggregate views" [1]).

B+-tree indexes play an important role in answering queries efficiently. 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 attributes of a view. However, in our study we consider only *fat indexes* over the lattice views — that is, those indexes that have a permutation of all of the grouping attributes of one of the views in the view lattice. We can extend our approaches to select among non-fat indexes as well. We plan to do this in our future work. Also, in our paper, we assumed that the attributes in `WHERE` clause and in `GROUP BY` clause are equally important. However, we can extend our approach to favor certain attributes to the others.

A SPJGA query q can be answered using a JGA view v only if the grouping attributes of v are a superset of the union of attributes in the GROUP BY clause of q and of the attributes in the WHERE clause of q that are compared to constants. By definition, each query q can be answered using the raw-data view in the lattice. 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.

2.1 Cost Model

The cost model that we use is similar to the one proposed in [11], i.e., the cost of answering query q using view v is the size of that portion of v that we must process in order to construct the result of q. When we answer query q using only view v with no indexes, we have to scan all rows of v to answer q. However, when we answer query q using view v and some index π on v, we read only the part of v referenced by the index with respect to the query. [11] generalizes the above observations to obtain a formula for the cost of answering query q using view v and index π.

Suppose A is the set of attributes in the `GROUP BY` clause of query q and the attributes that are compared with constant in the `WHERE` clause of query q. Also, suppose B is the set of grouping attributes of view v. If view v can answer query q, we have $A \subseteq B$. Furthermore, let $\vec{R}$ be an ordered set of attributes in index π over view v. $\vec{R}$ is a sequence of attributes of B. $\vec{R} = ()$ (the empty sequence) denotes the case where we are not using any index.

Let D denote the largest subset of A such that the attributes in D form a prefix (not necessarily proper) of $\vec{R}$. Then $C_q(\pi, v)$ which is the cost of answering query

q using view v and index π_v is defined as follows:

$$C_q(\pi, v) = \frac{size(v)}{size(v_D)}$$

where v_D is a view which its grouping attributes are the attributes in the set D. Note that $v_D = \phi$ represents the view which is aggregated on all of the attributes of the database.

We use sampling and analytical methods to compute the sizes of views in the view lattice. The size of a given view is the size (in bytes) of the distinct values of the attributes that the view groups by. Thus if the set of the `GROUP BY` attributes of view v_1 is a subset of the set of the `GROUP BY` attributes of view v_2, then $size(v_1) \leq size(v_2)$.

A typical cost measure for query-evaluation efficiency is the sum of the costs of evaluating the OLAP queries of interest, where the cost of each individual query in the sum may be weighted according to the frequency or importance of the query.

2.2 Problem Statement

In practical settings, the amount of available storage (disk) space is a typical natural *optimization constraint* in the (OLAP) view- and index-selection problem, as storing all possibly beneficial views and indexes is infeasible in today's database systems [2, 11]. This is still true even when we restrict our consideration to a set of frequent and important data-analysis queries instead of making the view- and index-materialization effort for all possible aggregate queries.

We consider the following OLAP view and index selection problem *OLAP-VI:* Given a star-schema data warehouse and a set of parameterized SPJGA queries, our goal is to minimize the evaluation costs of the queries in the workload, by selecting and precomputing (i) a set of lattice (JGA) views that can be used in answering the queries, and (ii) some fat indexes over those views. We consider this minimization problem under the 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 are of the form $I = (\mathcal{D}, \mathcal{Q}, b)$, where D is a database, Q is a workload of parameterized queries, and b is the (positive integer) value of the storage limit.

Definition. For a problem input $\mathcal{I} = (\mathcal{D}, \mathcal{Q}, b)$, a set of views and indexes $\mathcal{VI}$ is *admissible* if (1) each query in $\mathcal{Q}$ can be rewritten using views in $\mathcal{VI}$, and (2) views and indexes in $\mathcal{VI}$ satisfy the storage limit b. □

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

Definition. (OLAP-VI problem) For a given problem input $\mathcal{I} = (\mathcal{D}, \mathcal{Q}, b)$, find an optimal set of views and indexes. A solution for a given instance of the view-and index-selection problem for OLAP (OLAP-VI) 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 of indexes over views in $\mathcal{V}$, Π, and an association between each element of $\mathcal{Q}$ and its corresponding element of $\mathcal{V}$ and Π. □

Our problem statement is a special case of that of [11]; at the same time, we consider the hardest version of the problem statement of [11], by including in the initial search space of views and indexes the entire view lattice and all the fat indexes on the lattice views. Also note that, even though we consider "fat" indexes only, a straightforward modification of our approach can produce indexes with any number of columns.

3. THE IP MODEL

In this section we propose an integer-programming (IP) model for our OLAP view- and index-selection problem OLAP-VI. This IP model is the starting point from which we derive improvements, each of which significantly reduces the number of variables and constraints in the model.

To prune the search space of views we use a modification of our approach in [3]; For each view v in the view lattice V and a given query workload Q, we define $Q(v)$ as the set of queries in Q that can be answered by v. We consider a view v to be in the search space of views if each attribute of v is an attribute of one of the queries in $Q(v)$. We experimentally show the effectiveness of this method for reducing the size of the search space of views in Section 7. We use two sets of binary variables in our IP model. The variables in the first set are in the form $y_{v\pi q}$, for all $v \in V'$, $\pi \in I_v$, and $q \in Q$. The value of $y_{v\pi q}$ is one if and only if view v along with index π over v is selected to answer query q. Note that π in $y_{v\pi q}$ can represent the empty set, for the case where only view v without any index is selected to answer query q. The variables in the second set are in the form $x_{v\pi}$, where $v \in V'$ and $\pi \in I_v$. If index π of view v is selected for materialization, then $x_{v\pi} = 1$. Also, if view v is selected we have $x_{v\phi} = 1$.

Our problem OLAP-VI can now be stated as the following IP model:

$$\min \sum_{v \in V'} \sum_{\pi \in I_v} \sum_{q \in Q(v)} C_q(\pi, v) y_{v\pi q} \tag{1}$$

subject to

$$\sum_{v \in V'} \sum_{\pi \in I_v} y_{v\pi q} = 1 \qquad \forall q \in Q(v) \tag{2}$$

$$\sum_{v \in V'} \sum_{\pi \in I_v} size(v) x_{v\pi} \leq b \tag{3}$$

$$y_{v\pi q} \leq x_{v\pi} \qquad v \in V', \pi \in I_v, q \in Q(v) \tag{4}$$

$$x_{v\pi} \leq x_{v\phi} \qquad v \in V', \forall \pi \in I_v \tag{5}$$

$$x_{1\phi} = 1 \tag{6}$$

$$y_{v\pi q}, x_{v\pi} \in \{0, 1\} \; v \in V', \pi \in I_v, \forall q \in Q(v) \tag{7}$$

In this model, $C_q(\pi, v)$ is the cost of answering query q using view v and index π.

The meaning of constraint (2) is that each query should be answered by exactly one view and either no index or one of the indexes of that view. Constraint (3) states that the total storage requirement for the selected views and indexes should not exceed the prespecified amount b [3] . Constraint (4) ensures that if a view and one (or none) of its indexes is used to answer a query, then the corresponding view and index must be materialized. Constraint (5) implies that if an index is selected, its corresponding view should be selected too. Constraint (6) states that the raw-data view is always selected. Finally, the meaning of constraint (7) is that the variables in the model are all binary.

Suppose the value of storage space b is set to be

$$b = size(\text{raw-data view}) + \alpha \times \left(\sum_{q \in Q} size(q)\right) .$$

If $\alpha < 0$, the problem is infeasible, since the available storage space is not sufficient for storing the raw-data view. (If $\alpha = 0$ then the problem is not challenging.) If $\alpha \geq 2$, then the best solution is to materialize the raw-data view, all the queries, and an optimal index per query; [4] the cost (i.e., the value of the objective function) would be the number of queries. Thus, for the view- and-index-selection problem OLAP-VI to be nontrivial, we need $0 < \alpha < 2$. Note that the cost of answering each query is at least 1. As a result, the number of the queries in the workload is a lower bound on the cost in this model.

4. THE IPP MODEL

The search space of indexes and views in our IP model (Section 3), i.e., the sets V' and I_v for all $v \in V$, can be very large for realistic-size instances of our problem OLAP-VI. For each view v, there are $|v|!$ fat indexes in the search space. In this section we propose an approach to significantly reduce the number of indexes to be considered for each view, while still retaining all indexes associated with the optimum solution. We make the observation that only some points along an index π lead to query cost decreases – the attributes between such points can be arbitrarily permuted. With the help of an auxiliary graph G we can formalize this insight and reduce the number of candidate indexes from $|v|!$ to $2^{|Q(v)|}$, a significant reduction, especially if only a few queries benefit from v. We begin with two illustrative examples.

EXAMPLE 4.1. *Consider view $v = \{a, b, c, d, e, f\}$ and queries $\mathtt{Q_1} = \{a, b, c, d, e\}$, $\mathtt{Q_2} = \{a, b, c, d\}$, $\mathtt{Q_3} = \{b, c, d\}$, and $\mathtt{Q_4} = \{b\}$. (We represent each query or view as a set of attributes in its* `GROUP BY` *and* `WHERE` *clauses, and each index as a sequence of attributes of the underlying view.) Notice that in this example $\mathtt{Q_4} \subset \mathtt{Q_3} \subset \mathtt{Q_2} \subset \mathtt{Q_1}$, i.e., the queries form a "chain". Now consider index $\pi^* = (b, d, c, a, e, f)$. This index has the attributes of each of $\mathtt{Q_1}$, $\mathtt{Q_2}$, $\mathtt{Q_3}$, and $\mathtt{Q_4}$ as its prefix. Thus, for each query $q \in \{\mathtt{Q_1}, \mathtt{Q_2}, \mathtt{Q_3}, \mathtt{Q_4}\}$, the evaluation cost $C_q(\pi^*)$ does not exceed $C_q(\pi)$, where π is any of the 6! indexes over view V.* □

EXAMPLE 4.2. *Consider view $v = \{a, b, c, d, e\}$; let queries $\mathtt{Q_1} = \{a, b, c, d\}$ and $\mathtt{Q_2} = \{c, d, e\}$ be the only queries in $Q(v)$. Unlike example 4.1, these queries do not form a chain, yet they both have attributes c and d. Thus, those indexes over view v whose first two attributes are c and d can reduce the cost of answering queries $\mathtt{Q_1}$ and $\mathtt{Q_2}$ by at least a factor of $size(\{c, d\})$. For an index to further reduce the cost of $\mathtt{Q_1}$ it has to have attribute a or b immediately after c and d. To further reduce the cost $\mathtt{Q_2}$, e must be next. Storing both of the indexes (c, d, a, b, e) and (c, d, e, a, b) is the best choice for minimizing the cost of answering queries $\mathtt{Q_1}$ and $\mathtt{Q_2}$ using view v. However, if the space limit is sufficient for only a single index, one of those two indexes is still the best choice. This means that the $5! = 120$ indexes of I_v can be replaced with only two choices while still retaining the optimal solution.* □

In what follows we explain how to identify a restricted set of indexes I for an arbitrary view $v \in V'$. For each view v, we build a digraph G. The nodes of digraph G are sets of attributes in v: the empty set ϕ, the universal set v, the attributes of a query q in $Q(v)$, or attributes that two or more queries have in common. There is an edge from w_1 to w_2 if $w_1 \subset w_2$, and there is no node $w \in G$ with $w_1 \subset w \subset w_2$. Note that G has a single source ϕ and a single sink v (all attributes of v).

In Ex. 4.1 the nodes of G are the sets ϕ, $\{b\}$, $\{b, c, d\}$, $\{a, b, c, d\}$, and $\{a, b, c, d, e\}$, and the edges form a path that includes all nodes from ϕ to $\{a, b, c, d, e\}$. In Ex. 4.2 the nodes are ϕ, $\{c, d\}$, $\{c, d, e\}$, $\{a, b, c, d\}$, and $\{a, b, c, d, e\}$. There are two source-sink paths — one going through $\{c, d, e\}$, the other through $\{a, b, c, d\}$. As these examples illustrate, there is a discernible relationship between paths of G and indexes in our restricted set I.

Definition. A path P in G that begins at the source is *related* to an index π if every node along P is the set of attributes in a prefix of π. We say that P *agrees with* a query q if every node w along P has $w \subseteq q$. □

Definition. We define *the cost associated with index π of view v* as the total cost of answering queries in $Q(v)$ using view v and index π. □

From here on, every use of the word *path* refers to a path that begins at the source of G. The definition also applies to all such paths that go only part way to the sink.

Consider the relationship between an arbitrary query q and a path P. We can identify a node w such that P agrees with q up to and including w, and fails to agree after that — note that w may be the last node on P, in which case all of P agrees with q. For any node z on path P starting at w, we know that $w \subseteq z$. By construction of G, we can also deduce that $q \cap z = w$. In other words, none of the attributes in $z \setminus w$ are relevant

[3] We assume (see [11, 13]) that the storage requirement for each fat index is the same as that for the underlying view.

[4] We assume (see [11, 13]) that the raw-data view (i.e., the top of the view lattice) is always in the solution.

to q. Thus the cost of answering query q using any index related to P is the same, since any index related to P only has those attributes of q as a prefix that are in w. This observation leads to the following lemma.

LEMMA 4.1. *If P is a source-sink path in G and P is related to two indexes π_1, π_2, then the costs associated with π_1 and π_2 are the same.* □

The conclusion is that the cost associated with an index depends only on the unique source-sink path that agrees with it. The indexes related to a source-sink path P form an equivalence class w.r.t. cost: any index from that class can be chosen.

We formalize this for the set of optimal indexes I^* for view v.

LEMMA 4.2. *Let π be an index in I^* for view v and G be the diagraph related to view v. Then there is an index $\pi' \in I_v$ that is related to a source-sink path in G and has at most the same cost as π with respect to every query in $Q(v)$.* □

Proof. Let $P(\pi)$ be the longest path of G that is related to π. Let w be the last node in $P(\pi)$ and let attributes in $\{a_1, \ldots, a_k\}$ be what remains of π after the attributes of w have been removed. Suppose the order of attributes in $\{a_1, \ldots, a_k\}$ after attributes of w in π is $(a_1, \ldots, a_k)$. Let w_i be a child node of w that has all of the attributes in $\{a_1, \ldots, a_i\}$. Also, suppose w_i is a node with the largest value of i. If $i = k$, then w_i is a sink node and $P(\pi)$ is a source-sink path. As a result, based on the Definition 4, π is related to a source-sink path and $\pi' = \pi$. So suppose $i \neq k$. Consider a source-sink path P that $P(\pi)$ and w_i are part of it. Note that $P(\pi)$ along with w_i forms a path on P. Furthermore, suppose index π' is related to path P.

We categorize the queries in $Q(v)$ into three groups (some groups may not contain any query): (1) those that do not have all of the attributes in w, (2) the query that its attributes are the attributes in w, and (3) those that do have all of the attributes of w and some more attributes. The cost of answering any query q in groups 1 and 2 using π is the same as the cost of answering q using π' since the order of the attributes of q that form a prefix of π is the same in π'. Any query q in group 3 has all of the attributes in w_i, otherwise a node with attributes $q \cap w_i$ would be between node w and w_i on P which is against the fact that w is a direct parent of w_i on G. Also, π does not have all of the attributes in w_i as a prefix otherwise w would not be the last node on $P(\pi)$, but as π' is related to P, it has all of the attributes of w_i as a prefix. Knowing the fact that any query q in group 3 has all of the attributes in w_i, and knowing that π does not have all of the attributes in w_i as a prefix but π' does, results in the following: the cost of answering any query q in category 3 using index π' is not more than the cost of answering q using index π. □

What this lemma is saying is that we do not give up any optimal indexes by restricting ourselves to those that correspond to source-sink paths.

We are now ready to define the **IPP** model as an integer programming model that differs from our **IP** model of Section 3 in that we use the set I'_v in place of I_v. The set I'_v is defined by considering all source-sink paths P in the graph G for v and choosing, for each path, one π related to it.

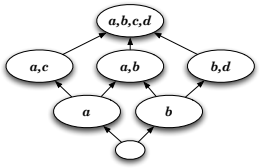


Figure 1: Graph G for Example 4.3.

The following is a direct consequence of Lemmas 4.1 and 4.2.

THEOREM 4.1. *Any optimal solution of the IPP model is optimal for the IP model.* □

EXAMPLE 4.3. *Consider view $\mathtt{V} = \{a, b, c, d\}$ and query workload $\mathtt{Q_V} = \{\mathtt{Q_1}, \mathtt{Q_2}, \mathtt{Q_3}, \mathtt{Q_4}\}$ where $\mathtt{Q_1} = \{a, b, c, d\}$, $\mathtt{Q_2} = \{a, c\}$, $\mathtt{Q_3} = \{a, b\}$, and $\mathtt{Q_4} = \{b, d\}$. Figure 1 represents graph G for $\mathtt{V}$. The paths in this graph are as follows:*

1. $\phi \rightarrow b \rightarrow b, d \rightarrow a, b, c, d$
2. $\phi \rightarrow b \rightarrow a, b \rightarrow a, b, c, d$
3. $\phi \rightarrow a \rightarrow a, b \rightarrow a, b, c, d$
4. $\phi \rightarrow a \rightarrow a, c \rightarrow a, b, c, d$

An index related to the first path should have first b, then d, and next a and c (a and c are in an arbitrary order at the end of the permutation). Thus index (b, d, c, a) is related to the first path. Indexes (b, a, c, d), (a, b, d, c), and (a, c, b, d) are related to the second, third, and forth paths, respectively. Thus we have:
$I'_v = \{(b, d, c, a), (b, a, c, d), (a, b, d, c), (a, c, b, d)\}$. □

A major limitation of the IPP model is the size of G, and hence the number of paths in it. In the worst case this will be exponential in the number of attributes. In the next two sections we will address this limitation by giving up potential optimal solutions in favor of decreasing the size of the search space of indexes.

5. THE IPN MODEL

Although our experiments show that our IPP approach (Section 4) is efficient in reducing the number of indexes considered in the search space of our IP model (Section 3), there are many realistic problem instances that we still cannot solve using our IPP model. Our next reduction in problem complexity comes about when we limit the number of paths to consider in the auxiliary graph G (Section 4). While this does not reduce the size of the graph, it will significantly reduce the number of variables and constraints in the model, thus reducing the time required to solve it. Except for a special case discussed at the end of this section, however, the smaller model may not yield an optimum solution to the overall problem.

First we introduce parameter $N(v)$, defined as an upper bound on the number of indexes required for view v to ensure a *locally optimal* solution with respect to v. By "locally optimal" we mean a solution that would be optimal if v were the only materialized view. As observed in the previous section, it suffices to choose one index per path in G — call this set of indexes $I_{N(v)}$. The IPN model is based on the choice of indexes yielded when $I_{N(v)}$ is chosen in place of I'_v. At the end of this section, we introduce a special case of our OLAP view- and index-selection problem, for which the IPN model guarantees an optimum solution.

Due to the space limit for storing indexes and the limited number of queries that each view can answer, there are upper bounds on $N(v)$. We have $N(v) \leq |Q(v)|$, since each query $q \in Q(v)$ can be answered optimally w.r.t. v by at most one index of v. Also, $N \leq \lfloor (b - size(v))/size(v) \rfloor$ because of the storage limit — each index requires $size(v)$ additional storage. Thus, $N = \min\{|Q(v)|, \lfloor (b - size(v))/size(v) \rfloor\}$.

Suppose we find a source-sink path P that yields the locally optimal solution when only one index is possible. Let $Q(P)$ be the set of queries q for which there exists a node w on P with $q \subseteq w$ — these are the queries that are helped by (indexes related to) P to the maximum extent possible. When there is room for more indexes, the locally optimum solution consists of an index related to P combined with the locally optimum solution over queries in $Q(v) \setminus Q(P)$.

The algorithm below finds the optimum path P in G. We can apply the algorithm $N(v)$ times. Each time we remove from G all nodes that exist only because of the queries in $Q(P)$.

Algorithm Optimal Path

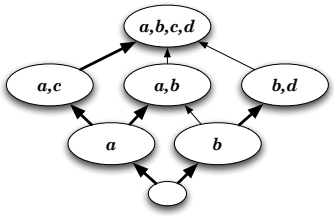
for each other node $w \neq \phi$ in G, in topological order do
  let $pred(w) = \{u \mid uw \text{ is an edge of } G\}$
  choose $u \in pred(w)$ with minimum $cost(path(u))$
  let $path(w) = path(u), w$
end do

At the end of Optimal Path, w is the sink and $path(w) = P$, the path we are looking for. The predecessor of w on P is the intersection of one or more queries, all of which agree with P.

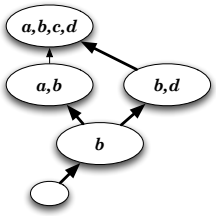
Given $N(v)$ we can use the graph G to compute the $N(v)$ best paths. We do $N(v)$ repetitions of algorithm Optimal Path. After a repetition computes a path P we update G, keeping the source, the sink, and all nodes that are intersections of queries in $Q(v) \setminus Q(P)$; all other nodes are discarded.

EXAMPLE 5.1. *Consider view* V *and the set of queries* Q(v) *described in Ex. 4.3. Suppose* N(v) = 2. *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 graph G is shown in Figure 2(a).*

Now, using $cost(w)$ as shorthand for $cost(path(w))$



(a) The original graph G.



(b) The modified graph after the first iteration.

Figure 2: Graphs for two iterations of Example 5.1.

the first repetition of the algorithm is

$$
\begin{aligned}
cost(\{a\}) &= 3 \times 1000/200 + 1000 = 1015 \\
&\quad a \in \mathtt{Q_1}, \mathtt{Q_2}, \mathtt{Q_3} \\
cost(\{b\}) &= 3 \times 1000/100 + 1000 = 1030 \\
&\quad b \in \mathtt{Q_1}, \mathtt{Q_3}, \mathtt{Q_4} \\
path(\{a,b\}) &= \{a\}, \{a,b\} \quad cost(\{a\}) < cost(\{b\}) \\
cost(\{a,b\}) &= 2 \times 1000/250 + 1000/200 + 1000 \\
&\quad = 1013 \quad a,b \in \mathtt{Q_1}, \mathtt{Q_3}; \quad a \in \mathtt{Q_2} \\
path(\{a,c\}) &= \{a\}, \{a,c\} \quad \textit{no other choice} \\
cost(\{a,c\}) &= 2 \times 1000/400 + 1000/200 + 1000 \\
&\quad = 1010 \quad a,c \in \mathtt{Q_1}, \mathtt{Q_2}; \quad a \in \mathtt{Q_3} \\
path(\{b,d\}) &= \{b\}, \{b,d\} \quad \textit{no other choice} \\
cost(\{b,d\}) &= 2 \times 1000/200 + 1000/100 + 1000 \\
&\quad = 1020 \quad b,d \in \mathtt{Q_1}, \mathtt{Q_4}; \quad b \in \mathtt{Q_3} \\
path(\{a,b,c,d\}) &= \{a\}, \{a,c\}, \{a,b,c,d\} \\
&\quad cost(\{a,c\}) < cost(\{a,b\}) \\
&\quad cost(\{a,b\}) < cost(\{b,d\}) \\
cost(\{a,b,c,d\}) &= 1000/1000 + 1000/400 + 1000/200 \\
&\quad +1000 \\
&= 1008.5 \\
&\quad a,b,c,d \in \mathtt{Q_1}; \quad a,c \in \mathtt{Q_2}; \quad a \in \mathtt{Q_3}
\end{aligned}
$$

At this point we can choose $\pi = (a,c,b,d)$, which relates to P. We see that $\mathtt{Q_1}$ *and* $\mathtt{Q_2}$ *are helped by π to the maximum extent possible. We remove node $\{a,c\}$ on path P which is associated with* $\mathtt{Q_2}$*, yet we keep node $\{a,b,c,d\}$ because it is a sink node. Furthermore, we remove node $\{a\}$ as it is not the intersection of those queries that are left in the graph, i. e. $\{a,b\}$ and $\{b,d\}$. The resulting graph is in Figure 2(b).* □

Special Case

In general, IPN does not guarantee that the solution it obtains is the optimum for the original problem. To see this note that $I_{N(v)}$ is the best set of $N(v)$ indexes over view v to answer queries in $Q(v)$, given view v is selected to answer all of the queries in $Q(v)$. But an optimum solution might not use v to answer all of the queries in $Q(v)$ – some queries could benefit more from other views.

However, if the set of selected views V^* in the optimum solution of IP model are such that no two views other than the raw-data view in V^* can answer the same query in the set Q and if no index is selected for the raw-data view, then any $v \in V^*$ must answer all of the queries in $Q(v)$. Considering the property of $I_{N(v)}$ discussed above, it follows that in this case any optimum solution of IPN is optimum for the original problem. Since $size(v)$ is always $\leq$ the size of the raw-data view, we can safely choose v instead of the raw-data view.

This property, in turn, guarantees that for certain instances of the view-and index-selection problem, the solution obtained by IPN is indeed guaranteed to be optimum for the original problem. In particular, we have the following special case:

THEOREM 5.1. *If in an instance of the view-and index-selection problem, the set of attributes of none of the queries in the workload is a subset of the union of the sets of attributes of other queries, then for this instance there exists a set of optimum views such that no two views other than the raw-data view in this set can answer the same query. In this case if the solution of IPN is in such a way that no index is selected for the raw-data view, IPN guarantees to provide the optimum solution for the original problem.* □

PROOF. Consider the set of optimum views for this instance. Suppose there exists a view v in this set that can answer query q, yet in the optimum solution, query q is selected to be answered by view $\hat{v}$ which is another view in the set of optimum views. In what follows, we explain how to substitute view v in the set of optimum solution by another view v' and how to substitute indexes over view v in the set of optimum indexes by some indexes over view v' such that 1) the total cost of evaluating queries does not increase, and 2) v' cannot answer q.

Consider view v' in the view lattice that its set of attributes is the union of the sets of attributes of queries in $Q(v) \setminus \{q\}$. Obviously, the set of attributes of v' is the subset of the set of attributes of v. Since the set of attributes of q is not a subset of the union of the sets of attributes of queries in $Q(v) \setminus \{q\}$, v' cannot answer q, but it can answer all other queries that are selected to be answered by v.

Suppose q' is a query that is selected to be answered by view v and index π_v over view v in the optimum solution. We have:

$$C_{q'}(\pi_v, v) = \frac{size(v)}{size(v_{q'})} \tag{8}$$

where $v_{q'}$ is the view that its attributes are the largest subset of attributes of q' that forms a prefix of π_v. Now consider the cost of answering query q' using view v' and index $\pi_{v'}$. Index $\pi_{v'}$ is an index over view v' related to index π_v where attributes in $\pi_{v'}$ have the same order as attributes in π_v, i.e., if attribute i is before attribute j in π_v, same is the order of i and j in $\pi_{v'}$. (Note that π_v has all of the attributes in $\pi_{v'}$). We have:

$$C_{q'}(\pi_{v'}, v') = \frac{size(v')}{size(v_{q'})} \tag{9}$$

Since v has all of the attributes in v' we have:

$$size(v') \leq size(v) \tag{10}$$

Also, since the order of attributes of q' is the same in π_v and $\pi_{v'}$ and π_v has more attributes than $\pi_{v'}$, the set of attributes of $v_{q'}$ is the subset of the set of attributes of $v'_{q'}$. Thus

$$size(v_{q'}) \leq size(v'_{q'}) \tag{11}$$

From equalities 8 and 9 and inequalities 10 and 11 we have:

$$C_{q'}(\pi_{v'}, v') \leq C_{q'}(\pi_v, v) \tag{12}$$

Since q' is an arbitrary query that is selected to be answered by v, inequality 12 is true for all queries that are selected to be answered by v. As a result, if we substitute v in the set of optimum views by v' and any index π_v over view v in the set of optimum indexes by its related index $\pi_{v'}$ over view v', the total cost of evaluating queries will not increase. Also, these substitutions do not violate any of the constraints of the problem.

We can repeat the above substitution procedure until there is no view other than $\hat{v}$ in the set of optimum views that can answer q. We also repeat this for all queries that can be answered by more than one view other than the raw-data view in the set of optimum views. This way we build a set of optimum views that no two view other than the raw-data view can answer the same query. □

6. THE IPNIR MODELS

In the previous section we achieved significant reduction in the size of the IP model — IPN has at most $|Q(v)|$ indexes to consider for each view v, instead of the potential $2^{|Q(v)|}$ of the IPP model of Section 4. To achieve this reduction, however, we still needed to create a graph with $2^{|Q(v)|}$ nodes in the worst case. Now we consider the possibility of creating only the most important part of the graph.

As we observed in Example 4.2, the order of the first attributes of each index are much more important than the order of the last attributes of that index. When the first 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 cost. A promising approach, therefore, is to restrict our construction of G to G_p, with only those nodes that represent intersections of at least p queries. The value of p can range from 1 to $|Q(v)|$. We also keep in G_p the nodes representing single queries in $Q(v)$. This is to ensure that we do not miss an opportunity to agree completely with a query.

In order to generate the $IPNIR(p)$ search space of indexes for each view v, we reuse the algorithm that

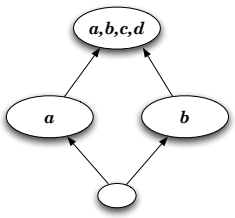


Figure 3: Graph G for Example 6.1.

generates $I_{N(v)}$, except that instead of using G in the first iteration, we use G_p. Note that for $p = 1$ or 2, G_p is the same as G. As we increase the value of p, the number of nodes in G_p will decrease; as a result, building the search space of indexes would be less time consuming. The size of G_p is $\sum_{i=1}^{k-p} c(k,i)$, where $k = |Q(v)|$ and $c(k,i)$ is the number of ways to choose i items from k. This is roughly $O(2^{k-p})$.

For some large instances, building the $IPNIR(p)$ search space of indexes for view v may require a lot of time even for large values of p. For this reason, we propose three other approaches to further reduce the number of nodes considered in G_p. In the first approach, we only consider nodes that represent queries and nodes that are immediate successors of the source (minimal number of attributes or maximal number of queries intersected without being empty) — call this $IPNIR$-QS.

In the second approach, we leave off the nodes that are queries, only considering the immediate successors of the source, intersections of as many queries as possible. We call this second approach $IPNIR$-S.

The other way to restrict the first approach gives us a final approach: only consider nodes that are queries. Call this approach $IPNIR$-Q.

EXAMPLE 6.1. *Figure 3 shows graph G_S in the first iteration of $IPNIR$-S.* □

Our experiments show that when the building time and solving time of $IPNIR(p)$ model is significantly different, the time needed to build $IPNIR(p)$ model is dominated by the time needed for CPLEX to solve the related integer programming model. See Section 7 for details of the experiments and analysis on $IPNIR(p)$ model.

Each of these approaches yields a corresponding integer programming model in the obvious way. In the remainder of the paper we examine the benefits of our various improvements (described in Sections 3-6) on our original IP model through experimental evidence.

7. EXPERIMENTAL RESULTS

In this section, we present the results of computational experiments to evaluate the performance of our proposed exact and inexact methods. The experiments consist of solving a collection of instances of the view-and-index-selection problem using each of the proposed algorithms and other competitive algorithms. For solving our instances, we have to choose a solver and a search space.

In our approaches, we used integer programming to select views and indexes. In order to solve the integer programming models, we used CPLEX [14] as a solver. We also used a greedy approach that we call it GHRU (defined below) to solve instances. Thus we used two solvers for solving our instances: CPLEX and GHRU.

The search space of views and indexes that we consider in our experiments are 1) the original search space of views and indexes which contains all of the views in the view lattice and all possible fat indexes for each view, 2) the IP search space of views and indexes where views are reduced to those that are the union of the queries that they can answer and for each such view, all possible fat indexes are considered in the search space of indexes, 3) IPP, 4) IPN, 5) IPNIR, and 6) ACN search space of views and indexes.

The results of our experiments show the followings:

- IPP is effective in reducing the size of the search space of views and indexes without removing any optimal view or index from the search space.
- Most of the time we obtain optimal solutions by applying CPLEX on IPN search space of views and indexes. Also, IPN search space is consistently a better search space than ACN in terms of including views and indexes that result in a lower cost of answering the queries, regardless of the solver that is used, i.e. CPLEX or GHRU. Moreover, the execution time for applying a solver on IPN and ACN search spaces are comparable.
- GHRU performs significantly better on IPN search space than any other search space mentioned above. In other words, GHRU by itself does not select a good combination of views and indexes, however if the search space is pruned before hand through IPN algorithm, then GHRU selects a much better combination of views and indexes.

In the remainder of this section we present a detailed account of our experiments and analysis.

7.1 Experimental Settings

We implemented 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. As mentioned earlier in this section, we used the CPLEX solver [14] to solve the integer-programming models. For comparative purposes we also independently developed computer programs for two other algorithms that we refer to as ACN and GHRU. We coded these algorithms in C++ as well and ran the code on the same platform.

The first algorithm we use in our experimental comparisons was proposed in [2]; its primary contribution is to reduce the size of the search space by constructing effective subsets of views and indexes. (See Section 1.1 for a detailed discussion of the contributions of [2].) For our experimental comparisons, we implemented an OLAP specialized version, which we call ACN, of that algorithm. In the first step of 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 and removes views v_1 and v_2 from the space. In this case, for each index of views v_1 and v_2, we also construct a similar index for the merged view v. [5] ACN terminates once it can add no more merged views to the search space.

Algorithm GHRU is the r-greedy algorithm proposed in [11] for selecting views and indexes for materialization, given a search space of views and indexes. [6] GHRU includes two types of basic steps: (i) select an index for an already selected view; or (ii) pick a view with at most $r-1$ selected indexes and enumerate all subsets of indexes choosing a set that maximizes the benefit per unit space. For a full description of the algorithm see [11].

In our experiments, we solved instances of the OLAP view- and index-selection problem OLAP-VI using different datasets of the TPC-H benchmark [23]. 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, 7, 9, 15]. In all of our instances, we set the value of storage space available to be equal to the size of the raw-data view plus half of the sum of the sizes of the queries, i.e. we set $\alpha = 0.5$.

7.2 Effectiveness of the View Reduction Algorithm

In this subsection, we study the effectiveness of our view reduction algorithm, i.e. we want to compare the size of the sets V and V'. Recall that V is the set of all views in the view lattice and V' includes those views in V that are the union of queries that they can answer. In other words, view v is in V' if each attribute of v belongs to at least one of the queries that v can answer.

To build V', first we compare the number of attributes in the database, k, with the number of queries in the workload, $|Q|$. Note that the number of views in the original search space of views is 2^k. If $k > |Q|$ then we build the set of views V' by considering each combination of r queries, $1 \leq r \leq |Q|$. Otherwise, we consider each view in the original search space of views and check whether it is the union of the queries that it can answer or not. As a result, the complexity of building the set V' is roughly $2^{min\{k,|Q|\}}$.

Our experiments include 10 instances over a 7-attribute TPC-H dataset. We add some queries to each instance to get the next instance. Each query in these instances has a random number of attributes between 1 and 6. In these instances, the size of the original search space of views is 127. Table 1 presents the number of views in set V' for each instance. Note that when the number of queries is relatively large (40 and 50), the size of the set V' is close to the size of the original search space of views, i.e. 127; however, when the number of queries is relatively small, the size of the set V' is much smaller than the size of the original search space of views.

[5] If π is an index of view v_1, we add the attributes in $v \setminus v_1$ to the end of index π and get an index for view v.

[6] For our experimental comparisons we chose, from the algorithms proposed in [11], an algorithm with the best performance guarantees, and set $r = 4$ for the parameter of this algorithm.

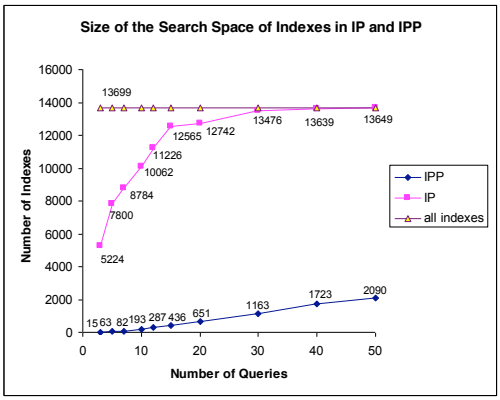


Figure 4: Number of indexes in IP and IPP for 10 instances on a 7-attribute TPC-H dataset.

no. of queries	no. of views
3	8
5	18
7	22
10	31
12	40
15	54
20	60
30	94
40	114
50	119

Table 1: Size of the reduced search space of views for instances over a 7-attribute TPC-H dataset.

7.3 Evaluating the Performance of IPP

Our next set of experiments is to evaluate the performance of IPP approach to reduce the search space of indexes. We solve ten instances over a 7-aqttribute TPC-H dataset [23]. Our instances are the same as instances in Subsection 7.2. For each instance, we measure the number of indexes in IP and IPP search spaces.

In these instances, the total number of indexes in the original search space is 13699. From Figure 4, we observe that the number of indexes in IPP search space of indexes is significantly smaller that the number of indexes in IP search space of indexes. Also, we can observe from this figure that for the instances with larger number of queries, the number of indexes in the search space of IP becomes closer to the number of indexes in the original search space. This is due to the larger number of views for instances with more queries.

For each of the instances described above, we measured the time required to build each of the IP and IPP search spaces and the time required to solve their related models with CPLEX. We observed that the time needed to build each of IP and IPP search spaces for larger instances (instances 4-10) is significantly smaller than the time needed to solve that instance using CPLEX. Table 2 shows the total time needed to build and solve each of the IP and IPP models for each instance. From this table we observed that the total time required to build IPP search space and apply CPLEX on it for each instance is significantly smaller than the total time required to build IP search space and apply CPLEX on it for that instance. In particular, applying CPLEX on the IP search space of views and indexes for the last seven instances took more than 15 minutes (our time limit), however, applying CPLEX on the IPP search space took one to eleven seconds for instances 4 to 9 and ninety eight seconds for the last instance.

of queries	IP time (sec)	IPP time (sec)
3	0.28	0.97
5	2.63	0.69
7	13.86	0.61
10	>15 min	1.14
12	>15 min	0.76
15	>15 min	1.23
20	>15 min	1.95
30	>15 min	15.13
40	>15 min	11.11
50	>15 min	98.45

Table 2: Comparison of the execution times for IP and IPP.

7.4 Impact of Parameter p on the Time Required for Building IPNIR Models

The purpose of the experiments in this subsection is to study the impact of parameter p in IPNIR on the number of indexes in the search space of indexes, building time of models, solving time, and the cost obtained by IPNIR.

We build 10 instances over a 13-attribute TPC-H dataset where each instance has 32 random queries and each query has a random number of attributes between 1 and 12. For each instance we compute the number of indexes in the search space of indexes, building time in seconds, solving time in seconds, and the cost obtained by IPNIR for p=2, 4, 8, 16, and 32.

Each graph in Figure 5 is related to an instance and shows the number of indexes for different values of p for that instance. From these graphs we observe that as the value of p increases, the number of indexes in the search space of indexes does not increase. The average difference between maximum and minimum number of indexes for each instance for different values of p is 9%.

The reason that sometimes the number of indexes in the search space decreases as we increase the value of p is that sometimes for some views like view v, the larger value of p results in smaller number of nodes in G_v and smaller number of source-sink paths for G_v. Thus, if N_v be large enough such that all indexes related to source-sink paths be in the search space, the size of the search space of indexes of v is smaller for larger values of p.

Each graph in Figure 6 is related to an instance and shows the time needed for building the IPNIR model for different values of p for that instance. We observe that as we increase the value of p, the building time never increases.

Each graph in Figure 7 is related to an instance and shows the time needed to solve the IPNIR model for different values of p for that instance. We do not observe any interesting pattern.

Each graph in Figure 8 is related to an instance and shows the cost obtained using IPNIR model for different values of p for that instance. Here again, we do not observe any interesting pattern.

Also, we observe that in almost all of our instances, the time needed for building the IPNIR model is significantly smaller than the time needed for solving that model, regardless of the value of p.

The results of these experiments are also presented in tables of Figure 9. In summary, we observe in our instances that the value of parameter p has a direct effect on the size of the search space of indexes and the time needed to build the model, but it does not affect the cost or the CPLEX time for solving the model in any special way.

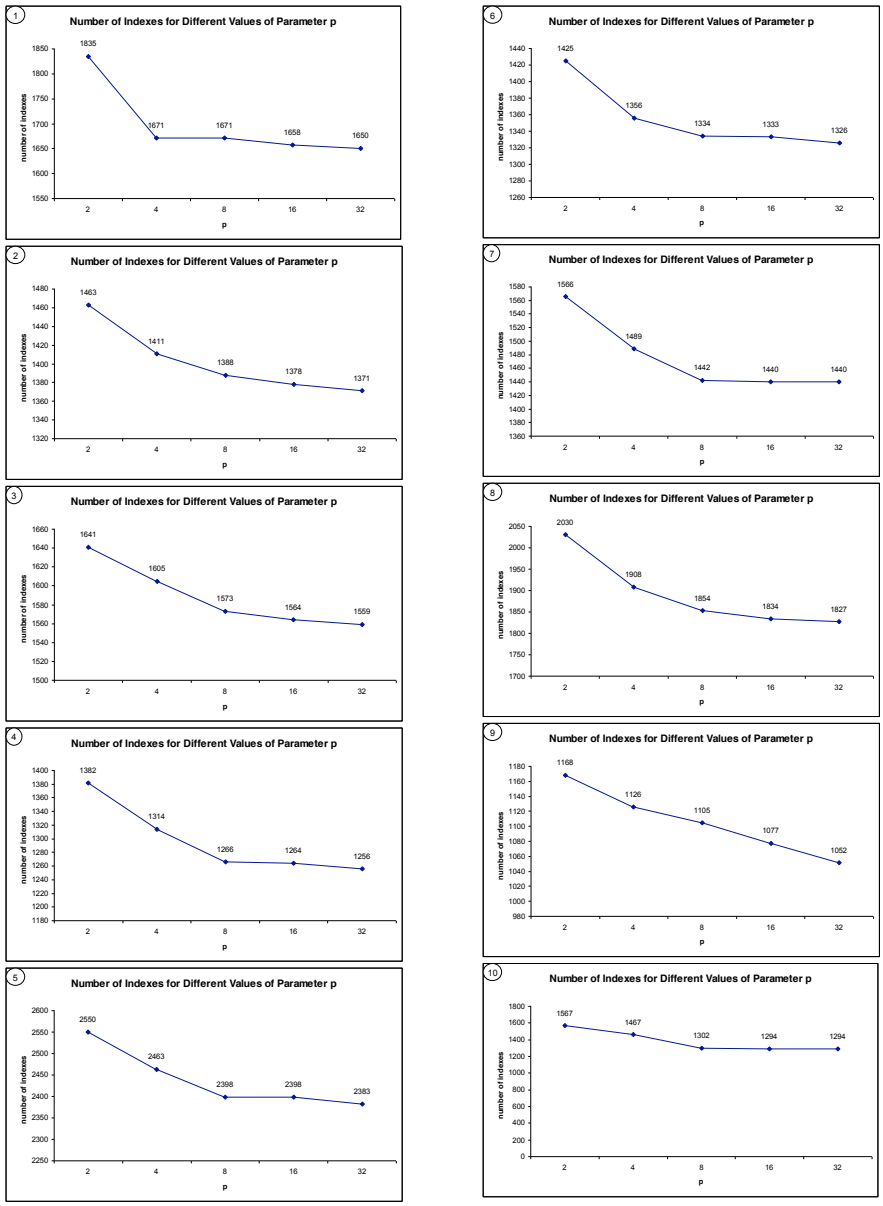


Figure 5: Effect of Parameter p on the size of the search space of indexes.

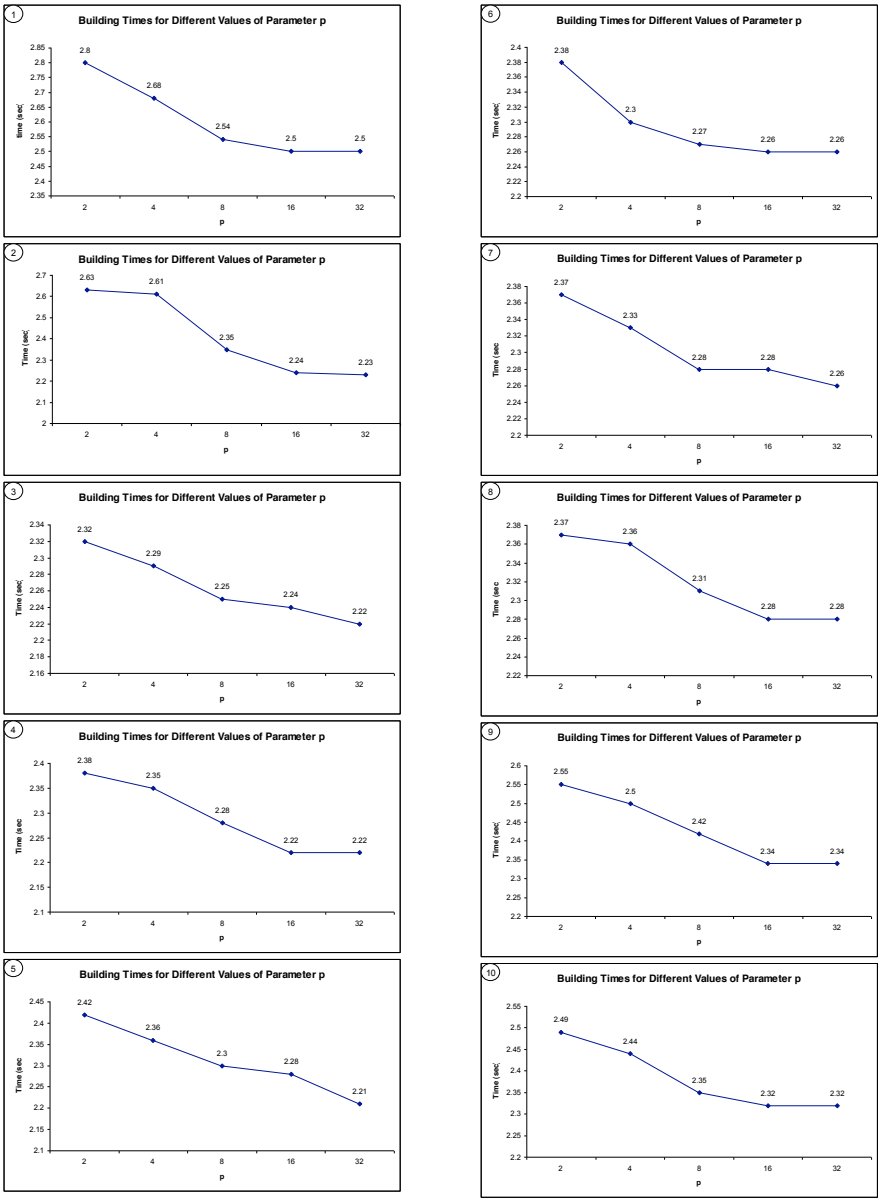


Figure 6: Effect of Parameter p on Time needed to Build IPNIR model.

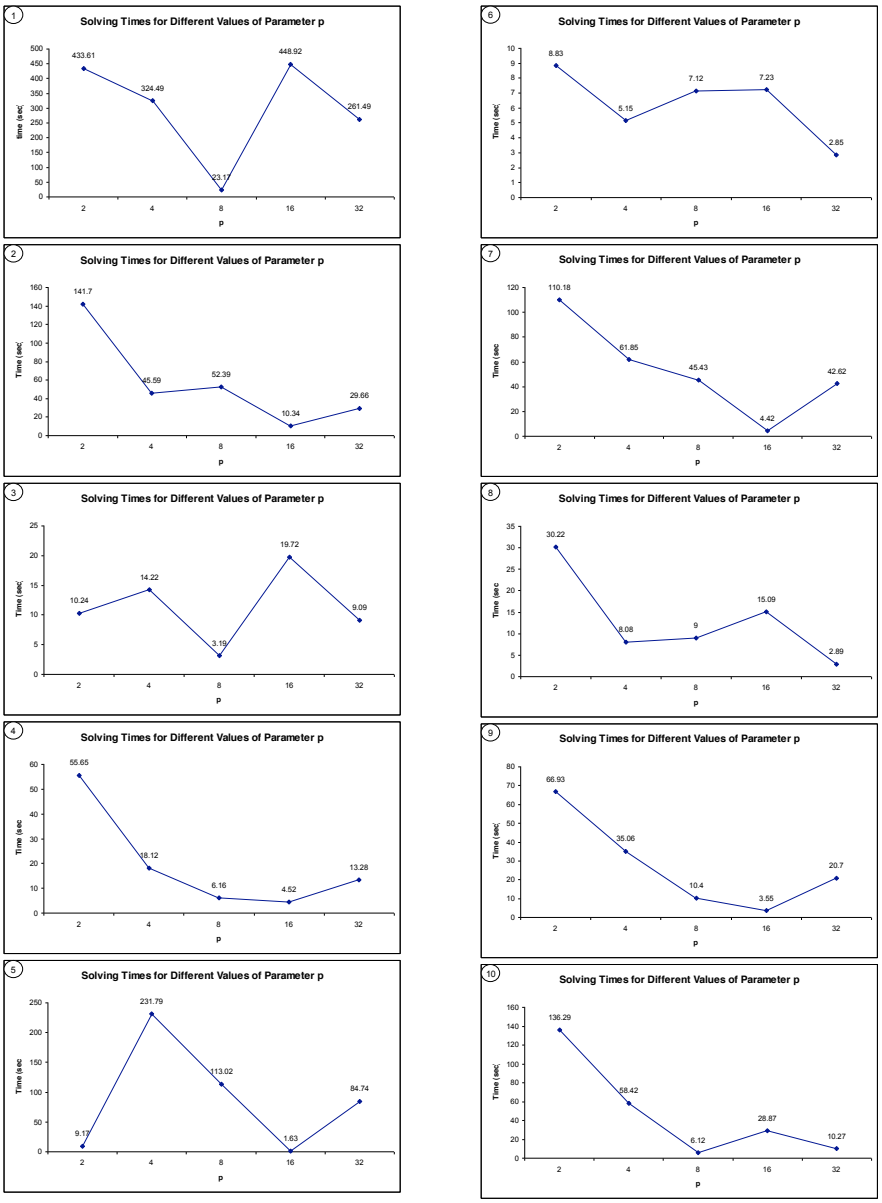


Figure 7: Effect of Parameter p on Time needed to Solve IPNIR model .

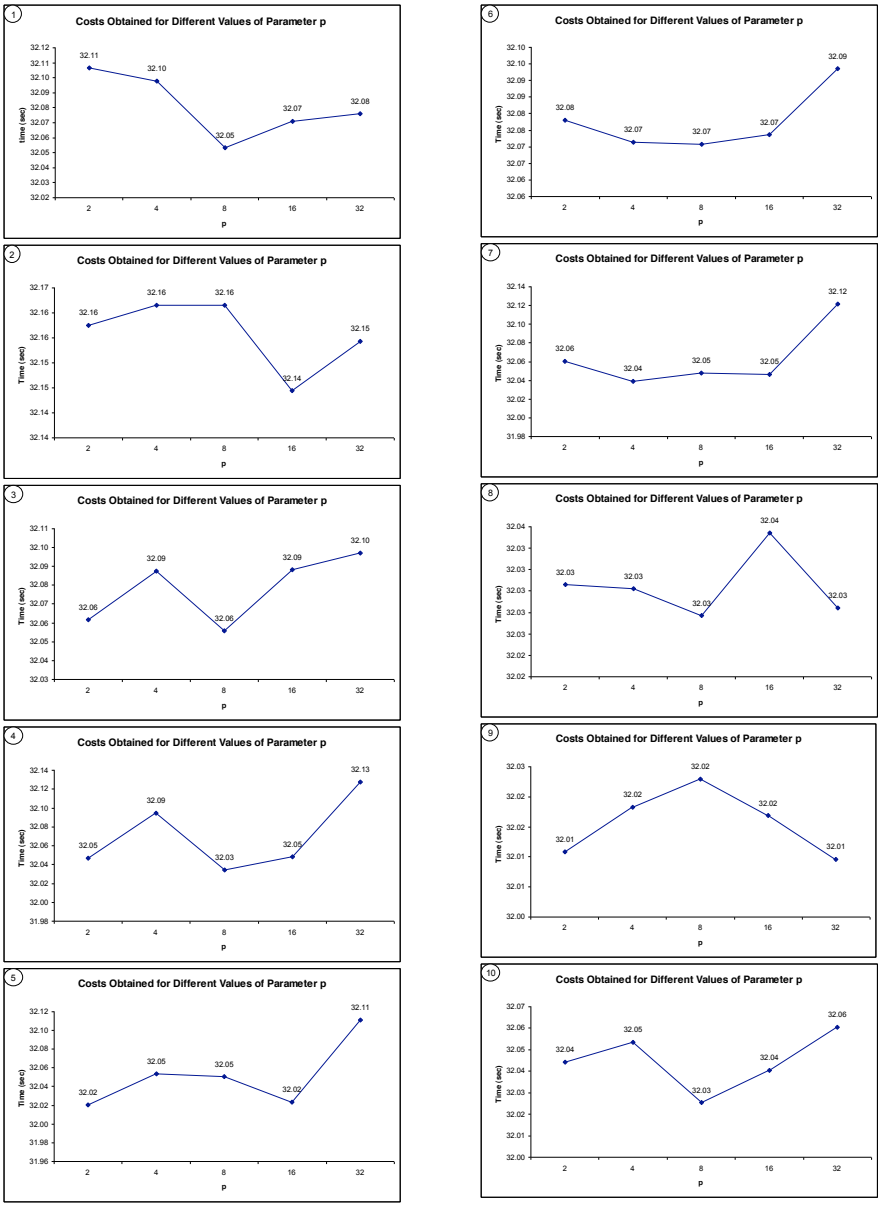


Figure 8: Effect of Parameter p on the Cost Obtained by IPNIR model.

instance 1

p	#indexes	BT (sec)	ST (sec)	cost
2	1835	2.80	433.61	32.11
4	1671	2.68	324.49	32.10
8	1671	2.54	23.17	32.05
16	1658	2.50	448.92	32.07
32	1650	2.50	261.49	32.08

instance 2

p	#indexes	BT (sec)	ST (sec)	cost
2	1463	2.63	141.7	32.16
4	1411	2.61	45.59	32.16
8	1388	2.35	52.39	32.16
16	1378	2.24	10.34	32.14
32	1371	2.23	29.66	32.15

instance 3

p	#indexes	BT (sec)	ST (sec)	cost
2	1641	2.32	10.24	32.06
4	1605	2.29	14.22	32.09
8	1573	2.25	3.19	32.06
16	1564	2.24	19.72	32.09
32	1559	2.22	9.09	32.10

instance 4

p	#indexes	BT (sec)	ST (sec)	cost
2	1382	2.38	55.65	32.05
4	1314	2.35	18.12	32.09
8	1266	2.28	6.16	32.03
16	1264	2.22	4.52	32.05
32	1256	2.22	13.28	32.13

instance 5 545

p	#indexes	BT (sec)	ST (sec)	cost
2	2550	2.42	9.17	32.02
4	2463	2.36	231.79	32.05
8	2398	2.30	113.02	32.05
16	2398	2.28	1.63	32.02
32	2383	2.21	84.74	32.11

instance 6

p	#indexes	BT (sec)	ST (sec)	cost
2	1425	2.38	8.83	32.08
4	1356	2.30	5.15	32.07
8	1334	2.27	7.12	32.07
16	1333	2.26	7.23	32.07
32	1326	2.26	2.85	32.09

instance 7

p	#indexes	BT (sec)	ST (sec)	cost
2	1566	2.37	110.18	32.06
4	1489	2.33	61.85	32.04
8	1442	2.28	45.43	32.05
16	1440	2.28	4.42	32.05
32	1440	2.26	42.62	32.12

instance 8

p	#indexes	BT (sec)	ST (sec)	cost
2	2030	2.37	30.22	32.03
4	1908	2.36	8.08	32.03
8	1854	2.31	9.00	32.03
16	1834	2.28	15.09	32.04
32	1827	2.28	2.89	32.03

instance 9

p	#indexes	BT (sec)	ST (sec)	cost
2	1168	2.55	66.93	32.01
4	1126	2.50	35.06	32.02
8	1105	2.42	10.40	32.02
16	1077	2.34	3.55	32.02
32	1052	2.34	20.70	32.01

instance 10

p	#indexes	BT (sec)	ST (sec)	cost
2	1567	2.49	136.29	32.04
4	1467	2.44	58.42	32.05
8	1302	2.35	6.12	32.03
16	1294	2.32	28.87	32.04
32	1294	2.32	10.27	32.06

Figure 9: Results of the Experiments in table format.

7.5 Evaluating the IPN Search Space of Views and Indexes

Our next set of experiments is to evaluate the performance of IPN by comparing it to the optimal approach IPP. To do so, we construct several instances over a 7 and a 13-attribute TPC-H datasets. For each instance, we build both IPP and IPN search spaces of views and indexes and apply CPLEX on each search space.

We solved seven instances over a 13-attribute TPC-H dataset; the first four instances have 10 random queries and instances five, six, and seven have 15, 15, and 20 random queries, respectively. Each query in these instances has a random number of attributes between 1 and 12.

We report the number of indexes in each of IPP and IPN search spaces for each instance in Table 3. From this table, we observe that the number of indexes in IPP is about 21 times more than the number of indexes in IPN.

instance	no. of queries	no. of indexes in IPP	no. of indexes in IPN
1	10	1218	60
2	10	409	52
3	10	534	55
4	10	358	52
5	15	5437	136
6	15	7069	181
7	20	15460	581

Table 3: Number of indexes in IPP and IPN search spaces for instances over a 13-attribute TPC-H dataset.

In Table 4 we report the total time needed to build each of the IPP and IPN search spaces and apply CPLEX on each search space for each instance. Also, we report the cost obtained from applying CPLEX on each search space. This table shows that except for the second instance, for all those instances that we could get the optimal solution in one hour (instances 1-6), the value of cost obtained from applying CPLEX on IPN is close to the optimal cost. Furthermore, the time needed to apply CPLEX on IPN is significantly smaller than the time needed to apply CPLEX on IPP for all of these instances. In particular, we observe that we were able to obtain a solution from applying CPLEX on IPN for a large instance (instance 7) in only about one second while CPLEX could not solve this instance in one hour when we applied it on IPP search space. Also, the cost that we obtained using IPN for this large instance, 21, is close to the lower bound for optimal cost in this instance, 20.

We also solved 11 instances over a 7-attribute TPC-H dataset where the number of queries in these instances varies from 20 to 50 and each query has a random number of attributes between 1 and 6. Similar to the previous set of experiments, we report the number of indexes in each of IPP and IPN search spaces for each instance in Table 5. From this table, we observe that the number of indexes in IPP is about 6 times more than the number of indexes in IPN.

instance	IPP cost (optimal)	IPN cost	IPP time (sec.)	IPN time (sec.)
1	10	10	2.10	0.73
2	10	20	0.96	0.75
3	10	10	0.96	0.74
4	10	10	0.83	0.72
5	15	15	287.34	0.95
6	15	15	520.74	0.99
7	-	21	>1hr	1.16

Table 4: Comparison of the cost obtained from applying CPLEX on IPP and IPN search spaces and execution times for instances over a 13-attribute TPC-H dataset

instance	no. of queries	no. of indexes in IPP	no. of indexes in IPN
1	20	913	117
2	20	500	108
3	20	561	144
4	20	521	114
5	20	713	112
6	20	692	98
7	20	694	98
8	20	418	109
9	30	802	202
10	40	1356	211
11	50	2117	287

Table 5: Number of indexes in IPP and IPN search spaces for instances over a 7-attribute TPC-H dataset

Similar to Table 4, in Table 6 we report the total time needed to build each search space and apply CPLEX on each search space for each instance. Also, we report the cost obtained from applying CPLEX on each search space. This table shows that except for the third and forth instances, for all of the instances, the value of cost obtained from applying CPLEX on IPN is close to the optimal cost. Furthermore, the time needed to apply CPLEX on IPN is significantly lower than the time needed to apply CPLEX on IPP for all of these instances. In particular, we observe that we were able to obtain a solution from applying CPLEX on IPN for a large instance (instance 11) in only about three seconds while CPLEX solve this instance in five hundred seconds when we applied it on IPP search space. Also, the cost that we obtained using IPN for this large instance is the same as the optimal cost. From Table 6 we also observe that except for two instances, the cost obtained via applying CPLEX on IPN is very close to the optimal cost.

instance	no. of queries	IPP cost (optimal)	IPN cost	IPP time (sec.)	IPN time (sec.)
1	20	20	20	10.14	0.81
2	20	20	22	2.17	0.77
3	20	20	53	3.66	0.9
4	20	20	74	1.93	0.95
5	20	20	20	3.26	0.81
6	20	20	20	4.49	0.81
7	20	20	21	2.99	0.77
8	20	21	25	4.01	1.04
9	30	30	30	13.82	1.23
10	40	40	40	50.57	1.16
11	50	50	50	500.5	3.07

Table 6: Comparison of the cost obtained from applying CPLEX on IPP and IPN search spaces and execution times for instances over a 7-attribute TPC-H dataset.

We did a preliminary investigation to find out why for some instances IPN provides a value for cost that is far from the optimal. Our investigation showed that small values for N(v) for some views which have large sizes and can answer many queries might be the reason. To clarify this, we present the following example:

EXAMPLE 7.1. *Suppose $V' = \{v_1, v_2\}$. Also suppose $Q(v_1) = \{q_1, q_2\}$ and $Q(v_2) = \{q_1, q_3\}$. Furthermore, assume N_{v_1}=N_{v_2}=1. So for each of views v_1 and v_2, only one index is in the search space of IPN. Suppose both of these indexes have all of the attributes of q_1 as their prefix. Also, suppose in the optimal solution, q_1 and q_3 are assigned to v_2 and q_2 is assigned to v_1. In this case, if the index over v_1 had instead of attributes of q_1 attributes of q_2 as its prefix, it could be more beneficial because v_1 is not selected for q_1.* □

We solved the instances that we did not get close to optimal solutions using IPN again by changing the value of $N(v)$ from $min\{\frac{b-s(v)}{s(v)}, |Q(v)|\}$ to $|Q(v)|$ for all views. Tables 7 and 8 show the results after we changed the value of N_v: the solutions that we obtain via IPN improves significantly; in fact we obtain optimal solutions after changing N_v to be $|Q(v)|$. Also in these three instances, although the number of indexes in the search space of IPN increases after changing N_v to be $|Q(v)|$, this number is still significantly smaller than the number of indexes in the search space of indexes of IPP. Furthermore, the time required for solving IPN after changing N_v did not significantly change for these three instances. We plan to work on determining a more proper definition for $N(v)$ in our future work.

7.6 Comparing with Other Heuristics

In this subsection, we compare the performance of our heuristic IPN with another heuristic ACN for reducing the size of the search space of views and indexes through several instances over two large size lattices.

Our first set of experiments consists of 10 instances over a 17-attribute TPC-H dataset. Each instance has

instance	no. of indexes in IPN before	no. of indexes in IPN after	no. of indexes in IPP
2 of the first set of experiments	52	77	534
3 of the second set of experiments	144	172	561
4 of the second set of experiments	114	159	521

Table 7: Comparing number of indexes in the search space of IPN before and after changing the definition of N_v for three instances that IPN provides solutions far from the optimal before changing the definition of N_v.

instance	IPN cost before	IPN cost after	optimal cost	IPN time (sec.) before	IPN time (sec.) after
2 of the first set of experiments	53	20	20	0.90	0.91
3 of the second set of experiments	74	20	20	0.95	0.90
4 of the second set of experiments	20	10	10	0.75	0.77

Table 8: Comparing the performance of IPN before and after changing the definition of N_v for three instances that IPN provides solutions far from the optimal before changing the definition of N_v.

20 random queries in the workload where each query has a random number of attributes between 1 and 16. For each instance, we first build IPN and ACN search spaces and then apply CPLEX on them. For all of these instances, the ACN search space of views and indexes is significantly smaller than the size of the IPN search space of views and indexes. On average, both the number of indexes and the number of views in ACN is two times smaller than the number of indexes and the number of views in IPN. Figure 10 compares the size of the search space of views in IPN and ACN for each instance and Figure 11 compares the size of the search space of indexes in IPN and ACN for each instance.

For each of these instances, we also measured the cost obtained from applying CPLEX on each of IPN and ACN search spaces and report it in Table 9. In this table, we also report the total execution time (building the search space and applying CPLEX) for solving each instance. From this table, we observe that the cost obtained via applying CPLEX on IPN for each instance is significantly smaller that the cost obtained via applying

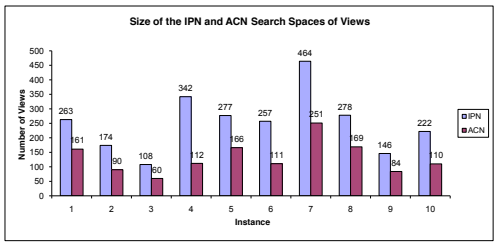


Figure 10: Comparison of the sizes of IPN and ACN search spaces of views for instances on 17-attribute

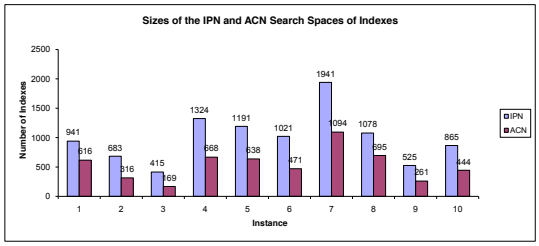


Figure 11: Comparison of the sizes of IPN and ACN search spaces of indexes for instances on 17-attribute TPC-H dataset.

CPLEX on ACN for that instance. Also, we observe that the total time needed to build IPN search space and apply CPLEX on it for each instance is comparable with the total time needed to build ACN search space and apply CPLEX on it for that instance.

The reason for the poor performance of algorithm ACN is that the order of the attributes of each index in the first iteration is a random order of attributes of a query. Thus most probably, each index is useful for one or only a few number of queries. Since the storage space is not enough to select one distinct index per query, there is no useful index among the selected indexes for some of the queries. To clarify this, we present the following example:

EXAMPLE 7.2. *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 5000, 2000, 2000, 10000, 15000, 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 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*).*

instance	IPN cost	ACN cost	IPN time (sec.)	ACN time (sec.)
1	20	359360	12.08	11.95
2	20	243476	1.22	10.43
3	20	92580	17.51	7.98
4	20	1810970	2.58	16.06
5	22	576012	2.44	32.20
6	20	144209	3.99	5.43
7	21	743439	11.02	40.76
8	20	35627	38.63	12.72
9	20	481878	6.94	4.28
10	20	43369	33.51	4.45

Table 9: Execution times and costs obtained via IPN and IPACN algorithms for instances on a 17-attribute TPC-H dataset. Each instance has 20 queries.

Since $s(v) \leq s(v1)+s(v2)$, *in the second iteration* v_1 *and* v_2 *will be substituted by* v *in the set of potential views and the set of potential indexes will be* $\{(b, a, c), (c, a, b)\}$. *Suppose the space limit allows us to only select one index over view* v. *Thus CPLEX chooses view* v *and index* (c, a, b) *which result in value 20001 for cost. However, the optimal index is* (a, c, b) *(not in the set of potential indexes of ACN) which results in value 5 for cost.* □

We also applied algorithm GHRU on IPN and ACN search spaces of each of the above instances. For each instance we measured the cost and the time needed to solve that instance with GHRU. In Table 10, we report the cost obtained via GHRU on each search space for each instance divided by the cost obtained via CPLEX on that particular search space for that instance. Also, we report the total time of building a search space and applying CPLEX on that search space for each instance divided by the total time of building a search space and applying GHRU on that particular search space for that instance.

From Table 10, we observe that the costs obtained via GHRU solver is 1 to 6.5 times more than the costs obtained via CPLEX solver, yet the execution time of GHRU is about 1 to 31 times smaller that the execution time of CPLEX.

Our second set of experiments consists of 10 instances over a 13-attribute TPC-H dataset. Each instance has 20 random queries in the workload where each query has a random number of attributes between 1 and 12. Again, for each instance, we first build IPN and ACN search spaces and then apply CPLEX on them. For all of these instances, the ACN search space of views and indexes is significantly smaller than the size of the IPN search space of views and indexes. On average, both the number of indexes and the number of views in ACN is two times smaller than the number of indexes and the number of views in IPN. Figure 12 compares the size of the search space of views in IPN and ACN for each instance and Figure 13 compares the size of the search space of indexes in IPN and ACN for each instance.

instance	$\frac{GHRU}{CPLEX}$ cost IPN	$\frac{GHRU}{CPLEX}$ cost ACN	$\frac{GHRU}{CPLEXtime}$ time IPN	$\frac{GHRUtime}{CPLEX}$ time ACN
1	1.0	1.0	0.10	0.31
2	1.2	1.4	0.86	0.34
3	1.0	6.5	0.06	0.44
4	1.2	1.0	0.66	0.23
5	5.5	1.1	0.47	0.11
6	1.0	2.9	0.27	0.65
7	1.9	1.2	0.16	0.09
8	1.1	2.0	0.03	0.28
9	1.0	1.6	0.15	0.84
10	1.3	2.0	0.03	0.82

Table 10: Comparing the costs obtained from applying CPLEX and GHRU on IPN and ACN search spaces and their execution times for instances over a 13-attribute TPC-H dataset. Each instance has 20 queries in the workload.

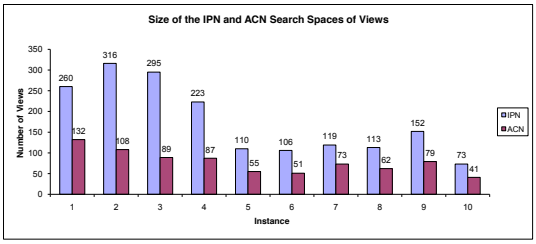


Figure 12: Comparison of the sizes of IPN and ACN search spaces of views for instances on 13-attribute TPC-H dataset.

For each of these instances, we also measured the cost obtained from applying CPLEX on each of IPN and ACN search spaces and report it in Table 11. In this table, we also report the total execution time (building the search space and applying CPLEX) for solving each instance. Our observations from this table is consistent with our observation in the previous set of experiments: the cost obtained via applying CPLEX on IPN for each instance is significantly smaller that the cost obtained via applying CPLEX on ACN for that instance. Also, the total time needed to build IPN search space and apply CPLEX on it for each instance is comparable with the total time needed to build ACN search space and apply CPLEX on it for that instance.

As in our previous set of experiments, we also applied algorithm GHRU on IPN and ACN search spaces of each of the above instances and present the result in Table 12. From this table, we observe that the costs obtained via GHRU solver is 1 to 3.9 times more than the costs obtained via CPLEX solver, yet the execution time of GHRU is about 1 to 32 times smaller that the

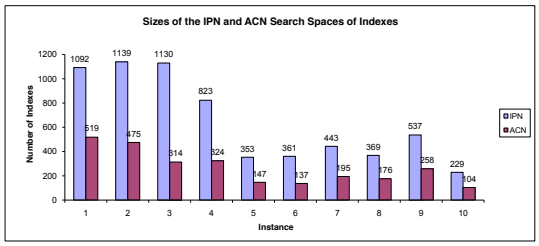


Figure 13: Comparison of the sizes of IPN and ACN search spaces of indexes for instances on 13-attribute TPC-H dataset.

instance	IPN cost	ACN cost	IPN time (sec.)	ACN time (sec.)
1	85	469959	28.59	7.72
2	25	769250	1.88	37.19
3	21	1237010	4.57	21.52
4	20	656023	5.18	13.08
5	20	598928	2.13	27.4
6	20	74779	4.17	5.6
7	20	72352	1.31	8.47
8	20	19959	1.8	15.58
9	20	336517	15.16	19.03
10	20	3839	3.4	3.98

Table 11: Execution times and costs obtained via IPN and IPACN algorithms for instances on a 13-attribute TPC-H dataset. Each instance has 20 queries.

execution time of CPLEX.

7.7 Evaluating the Performance of GHRU on Different Search Spaces of Views and Indexes

In this subsection, we study the performance of GHRU on five different search spaces of views and indexes mentioned at the beginning of this section. To do so, We solved six instances over a 7-attribute TPC-H dataset using GHRU solver. For each instance, we applied GHRU on 1) the original, 2) IP, 3) IPP, 4) IPN, and 5) ACN search spaces. We used a small lattice (7-attribute) so that we will be able to solve GHRU on the original search space as well. Each of our instances has 10 random queries in the query workload where each query has a random number of attributes between 1 and 6. For each instance, we measured the cost obtained via GHRU on each search space. We observe that in all of our instances, the performance of GHRU on IPN search space results in the smallest cost. Thus in Table 13 for each instance we report the cost obtained via GHRU on each search space divided by the cost obtained via

instance	$\frac{GHRU}{CPLEX}$ cost IPN	$\frac{GHRU}{CPLEX}$ cost ACN	$\frac{GHRU}{CPLEXtime}$ time IPN	$\frac{GHRUtime}{CPLEX}$ time ACN
1	1.7	1.0	0.03	0.47
2	1.2	1.6	0.51	0.10
3	3.9	1.2	0.22	0.17
4	1.0	1.1	0.17	0.28
5	1.0	1.2	0.41	0.14
6	1.0	1.1	0.20	0.64
7	1.5	1.8	0.69	0.43
8	1.0	0.7	0.48	0.24
9	1.0	1.0	0.06	0.19
10	1.1	1.1	0.27	0.91

Table 12: Comparing the costs obtained from applying CPLEX and GHRU on IPN and ACN search spaces and their execution times for instances over a 13-attribute TPC-H dataset. Each instance has 20 queries in the workload.

GHRU on IPN search space for that instance. From this table, we observe that GHRU performs better on IPP search space than the original, IP, and ACN search spaces. The performance of GHRU on the original, IP, and ACN search spaces are comparable.

For each instance we also measured the time required to solve that instance using GHRU and report it in Table 14. From this table we observe that on average the time required to solve instances by applying GHRU on the original, IP, IPP, IPN, and ACN search spaces for each instance is 615.50, 112.56, 12.63, <0.01, and 0.02 seconds, respectively. Thus, not only we obtained the best cost when we applied GHRU on IPN search space, but also applying GHRU on IPN is faster than applying GHRU on any other search space.

instance	$\frac{original}{IPN}$ GHRU cost	$\frac{IP}{IPN}$ GHRU cost	$\frac{IPP}{IPN}$ GHRU cost	$\frac{ACN}{IPN}$ GHRU cost
1	73351	73351	1	87290
2	4	4	1	3
3	20471	20492	2	3
4	27890	27949	20917	27890
5	92251	92251	23064	81646
6	4	4	2	2

Table 13: Comparing the quality of the solutions obtained via GHRU on different search spaces.

Our above observations show that for the instances we solved, GHRU performs better on smaller size search spaces IPN and IPP, rather than larger size search spaces, IPP and original. This means that most of the time GHRU by itself does not select a good combination of views and indexes, however, if the search space is pruned before hand by our approaches, 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 due to the fact that most of the times ACN does not contain good combination of views and indexes.

instance	GHRU on original time (sec.)	GHRU on IP time (sec.)	GHRU on IPP time (sec.)	GHRU on IPN time (sec.)	GHRU on ACN time (sec.)
1	817.78	158.1	38.13	<0.01	<0.01
2	524.42	111.28	17.33	<0.01	0.03
3	614.86	95.59	15.07	<0.01	0.04
4	641.01	105.43	0.01	<0.01	<0.01
5	571.1	137.56	5.25	<0.01	<0.01
6	523.84	67.37	0.01	<0.01	0.04

Table 14: Comparing the solving time of GHRU on different search spaces.

8. CONCLUSIONS

In this paper we undertook a systematic study of the OLAP view- and index-selection problem, and proposed a family of algorithms that effectively and efficiently prune the space of potentially beneficial views and indexes. Our experiments show that our proposed approaches to view and index selection result in high-quality solutions — in fact, in *globally optimum* solutions for many realistic-size problem instances. Thus, they compare favorably with the well-known OLAP-centered approach of [11] and provide for a winning combination with the end-to-end framework of [2] for generic view and index selection.

This project, together with our other work [18, 19], lays the foundation for studying view and index selection in a systematic principled way. (The project reported in this paper builds on our systematic studies [3, 19] of the OLAP view-selection problem.) 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 optimum*) counterparts, rather than just with respect to the base line where the system does not use any views. Directions of our current and future work in this project include finding special cases of practical significance where good approximability of the OLAP view- and index-selection problem can be achieved.

9. ACKNOWLEDGMENTS

The authors' work has been partially supported by NSF grants DMI-0321635, 0307072, and 0447742.

10. REFERENCES

[1] F. N. Afrati and R. Chirkova. Selecting and using views to compute aggregate queries (extended abstract). In *ICDT*, pages 383–397, 2005.

[2] S. Agrawal, S. Chaudhuri, and V. R. Narasayya. Automated selection of materialized views and indexes in SQL databases. In *VLDB*, pages 496–505, 2000.

[3] Z. Asgharzadeh Talebi, R. Chirkova, and Y. Fathi. Exact and inexact methods for solving the

problem of view selection for aggregate queries. Technical Report TR-2007-27, NC State University, 2007.

[4] E. Baralis, S. Paraboschi, and E. Teniente. Materialized view selection in a multidimensional database. In *Proc. VLDB*, pages 156–165, 1997.

[5] 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.

[6] 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.

[7] S. Chaudhuri, M. Datar, and V. R. Narasayya. Index selection for databases: A hardness study and a principled heuristic solution. *IEEE Trans. Knowl. Data Eng.*, 16(11):1313–1323, 2004.

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

[9] S. Chaudhuri and V. R. Narasayya. An efficient cost-driven index selection tool for microsoft SQL server. In *VLDB*, pages 146–155, 1997.

[10] C. I. Ezeife. A uniform approach for selecting views and indexes in a data warehouse. In *IDEAS*, pages 151–160, 1997.

[11] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *ICDE*, pages 208–219, 1997.

[12] H. Gupta and I. S. Mumick. Selection of views to materialize in a data warehouse. *IEEE Trans. Knowl. Data Eng.*, 17(1):24–43, 2005.

[13] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD Conference*, pages 205–216, 1996.

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

[15] P. Kalnis, N. Mamoulis, and D. Papadias. View selection using randomized search. *Data Knowledge Engineering*, 42(1):89–111, 2002.

[16] H. J. Karloff and M. Mihail. On the complexity of the view-selection problem. In *PODS*, pages 167–173, 1999.

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

[18] M. Kormilitsin, R. Chirkova, Y. Fathi, and M. Stallmann. View and index selection for query-performance improvement: Algorithms, heuristics and complexity. Technical report, NC State University, 2007.

[19] J. Li, Z. A. Talebi, R. Chirkova, and Y. Fathi. A formal model for the problem of view selection for aggregate queries. In *ADBIS*, pages 125–138, 2005.

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

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

[22] A. Shukla, P. Deshpande, and J. F. Naughton. Materialized view selection for multidimensional datasets. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 488–499, 1998.

[23] TPC-H:. TPC Benchmark H (Decision Support). Available from `http://www.tpc.org/tpch/spec/tpch2.1.0.pdf`.

[24] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proc. VLDB*, pages 136–145, 1997.