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

(extended abstract)

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 help to some extent 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 [17] 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 when 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 problem instances, although we no longer guarantee the global optimality of the resulting solution. (4) Finally, we present an experimental comparison of our proposed approaches with the state-of-the-art approaches of [2, 12]. Our experiments show that our 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 [12] 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 [9]. 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.

The prominent role of materialized views and indexes in improving query-processing performance has long been recognized, see, for instance, [6, 21]. Enterprise-class database-management systems that include modules for *generic* view and index selection include Microsoft SQL

Server [2, 22] and DB2 [6]. 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 [19], 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 [12] 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 formally defined the OLAP view- and index-selection problem. We begin by giving here an informal discussion of this optimization problem. We provide a detailed discussion in Section 2; a precise definition of the OLAP view- and index-selection problem is available in [4].[1]

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- and index-selection problem under the storage-space constraint. 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 [12]), 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 [12]. Thus, it is natural to look for heuristic solutions. Well-known past efforts in this direction include [2, 12]; 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 [17] 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 in-

---

[1] All the details that are omitted in this paper due to the space limit can be found in our technical report [4], which is available online.

dexes, with the hope that the resulting search space is small enough, for practical problem instances, that we can use software such as CPLEX [15] to get optimum or 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 [15].

- 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 [12] with the performance of all IP-based approaches that we propose here. Of course, whenever 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 [12]), 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 [12] is not possible. In section 7 we report the results of applying our IP-based heuristic and the approach of [12] 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 [12, 13, 14]. In particular, a well-known paper [12] 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 [17] disproved the strong performance bounds of these algorithms, by showing that the underlying approach of [14] cannot provide the stated worst-case performance ratios unless P=NP. As observed in [17] 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.

[11] discusses a uniform approach for selecting views and indexes for OLAP queries. This approach consid-

ers 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 [11] leaves out most implementation detail as well as any performance study of the proposed approach, which makes the approach 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 [10] 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 queries, by providing the user with solution-quality guarantees on the views and indexes to be materialized. Also see [19] 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., [7, 8] and references therein) or selection of views only (see, e.g., [5, 16, 23, 25] and references therein) for OLAP. In particular, Yang and colleagues [25] 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 [9, 18]. Similarly to [12, 14, 16, 23], 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 dimen-

sion 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 [14], which includes all star-join views with grouping and aggregation (JGA views) on the base tables. Each lattice view (1) has grouping on some of the attributes 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 the work reported here 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. It is straightforward to extend our approaches to select among other types of indexes as well. Also, we assume here that the attributes in the `WHERE` clause and in the `GROUP BY` clause of the queries are equally important. Again, our approaches can be extended to favor certain attributes over 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 the attributes in the GROUP BY clause of q and of those attributes in the WHERE clause of q that are compared to constants. By definition, each query q can be answered using the top (i.e., *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 [12]. 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_v on v, we read only the part of v referenced by the index with respect to the query. We use the standard cost measure for query-evaluation efficiency, namely 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, 12]. 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 storing the materialized views and indexes.

Our problem statement is a special case of that of [12]; at the same time, we consider the hardest version of the problem statement of [12], by including in the initial search space of views and indexes the entire view lattice and all the fat indexes on the lattice views. (As has been observed above, 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, see Sections 4–6.

To prune the search space of views we use a modification of our approach in [3]; please refer to [3, 4] for the details on view selection in the approach proposed in this paper. 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 [3].

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:

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

subject to

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

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

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

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

$$x_{1\phi} = 1 \qquad (6)$$

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

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.[2] 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;[3] 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. In particular, 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 an optimum solution. We make the observation that only some "points along" (the attribute permutation for) 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 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 the exposition 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, i.e., permutation, 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. $\square$

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 the queries of 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 an optimal solution.* $\square$

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 (i) $w_1 \subset w_2$, and (ii) 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 Example 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 Example 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 $w \subseteq q$. $\square$

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. $\square$

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 paths going 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 to q. Thus, the cost of answering query q using any index related to P is the same, since any index related to P has as a prefix only those attributes of q 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 of "path related to index", π is related to a source-sink path and $\pi' = \pi$. So suppose $i \neq k$. Consider a source-sink path P that includes $P(\pi)$ and w_i. 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 be empty): (1) those that do not have all of the attributes in w, (2) the queries whose attributes are exactly the attributes in w, and (3) those that have all of the attributes of w and other 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 contradicts 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, leads us to conclude the following: the cost of answering any query q in group 3 using index π' does not exceed the cost of answering q using index π. □
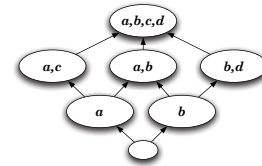

Figure 1: Graph G for Example 4.3.

What Lemma 4.2 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.

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

1. $\phi \to b \to b, d \to a, b, c, d$

2. $\phi \to b \to a, b \to a, b, c, d$

3. $\phi \to a \to a, b \to a, b, c, d$

4. $\phi \to a \to a, c \to a, b, c, d$

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 Sections 5–6 we 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 $I_{N(v)}$. Our 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, we have $N(v) \leq |Q(v)|$. (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 node $w \neq \phi$ in G, in topological order **do**
    let $pred(w) = \{u \mid uw$ 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 Example 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).*
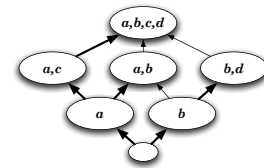
(a) The original graph G.
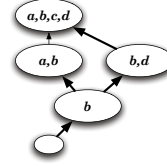
(b) The modified graph after the first iteration.

Figure 2: Graphs for two iterations of Example 5.1.

Now, using cost(w) *as shorthand for* cost(path(w)) *the first repetition of the algorithm is*

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

At this point we can choose $\pi = (a, c, b, d)$, *which relates to* P. *We see that* Q₁ *and* Q₂ *are helped by* π *to the maximum extent possible. We remove node* $\{a,c\}$ *on path* P *which is associated with* Q₂, *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 optimal for the original problem. To see this, observe that $I_{N(v)}$ is the best set of $N(v)$ indexes over view v to answer queries in $Q(v)$, provided view

v is selected to answer all 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 views V^* output in an optimum solution of our IP model is 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 optimal solution of IPN is optimal for the original problem. Since $size(v)$ never exceeds 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 optimal for the original problem. In particular, if in an instance of the problem, none of the queries is a subset of the union of other queries, the set of optimal views has the above property and IPN is guaranteed to provide an optimal solution for the original problem. (See [4] for a formal proof of this claim.)

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 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
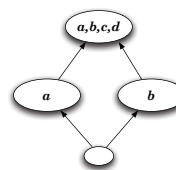

Figure 3: Graph G for Example 6.1.

that are immediate successors of the source (minimal number of attributes or maximal number of queries intersected without being empty) — call this $IPNIR\text{-}QS$.

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 $IPNIR\text{-}S$.

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

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

Our experiments show that when the building time of the $IPNIR(p)$ model differs significantly from its solving time, the time needed to build the $IPNIR(p)$ model is dominated by the time needed for CPLEX to solve the related integer-programming model. See [4] for the details of the experiments and analysis for the $IPNIR(p)$ 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 a computational experiment to evaluate the effectiveness of our proposed exact and inexact algorithms. The experiment consists of solving a collection of instances of the view- and index-selection problem using each of the proposed algorithms and other competitive algorithms. These results show that our heuristic procedure IPN and its variants consistently outperform other heuristic procedures in terms of the quality of solutions obtained, although their corresponding execution time is moderately larger. More specifically, our experimental results show that:

1. For smaller instances of the problem, our exact methods IP and IPP are both able to solve the problem optimally, with IPP having a clear advantage over IP in terms of both its execution time and memory requirements, due to its reduced search space. For larger instances, however, even for IPP the memory requirements and execution time can be excessively large.

2. Among the heuristic (inexact) algorithms with reduced search space, i.e., our proposed algorithm

IPN and several variants of two state-of-the-art algorithms that we refer to as ACN and GHRU (defined below), our proposed algorithm IPN consistently finds solutions with better (lower) objective-function values than the other methods. In several of the instances in our experiment, the values of the solutions obtained by IPN are, in fact, quite close to the respective optimal values.

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. We used the CPLEX solver [15] to solve the integer-programming models. For comparative purposes we also independently developed computer programs for two other algorithms, which 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 that 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 discussion of the contributions of [2].) For our experiments 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 candidate view, and a randomly selected order of the attributes of each such view is considered as a candidate index for that view. Subsequently, ACN considers the union of each pair of views, v1 and v2, as a new "merged" view v. If the size of a merged view v is less than the sum of the sizes of views v1 and v2, the algorithm adds view v to the search space of views and removes views v1 and v2 from the space. In this case, for each index of views v1 and v2, we also construct a similar index for the merged view v.[4] ACN terminates once it can add no more merged views to the search space.

Algorithm GHRU is the r-greedy algorithm proposed in [12] for selecting views and indexes for materialization, given a search space of views and indexes.[5] 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 the indexes to choose a set that maximizes the benefit per unit space. (See [12] for the details.)

We solved several instances of the OLAP view- and index-selection problem OLAP-VI using different datasets of the TPC-H benchmark [24]. The sizes of the instances that we solved are realistic and comparable to the sizes of the instances used in related work (cf. [2, 8, 10, 16]). For each instance, we first built the complete search space of views and indexes and then applied the

---

[4]If π is an index of view v1, we add the attributes in v\v1 to the end of index π and get an index for view v.

[5]For our experimental comparisons we chose, from the algorithms proposed in [12], an algorithm with the best performance guarantees.
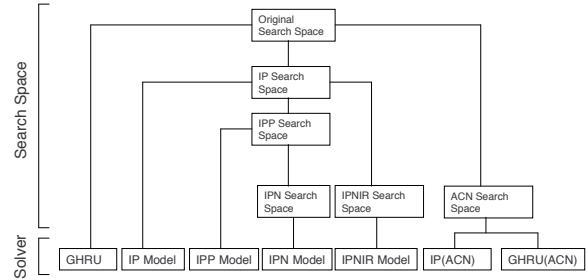

Figure 4: The search spaces and algorithms in our experiments.

relevant algorithms. For each instance we also built the ACN search space and applied the IP and GHRU algorithms on this search space. Figure 4 illustrates the search spaces and the algorithms that we applied on each search space.

Table 1 presents the results of our experiments over four representative instances of the view- and index-selection problem OLAP-VI.[6] Each row in this table corresponds to an instance of the problem and a specific algorithm applied to that instance, as indicated in the first two columns of the table. The remaining columns of the table contain pertinent information about the instance, as well as the results obtained by the corresponding algorithm, as stated in the column headers. The total execution time (in seconds) is reported in the last column of the table. The preceding two columns give a more detail account of this execution time, by separating it into two components. The "building time" (column 6) is the amount of time required to determine the search space, and the "solving time" (column 7) is the amount of time required to select the final set of views and indexes, i.e., the solution to the problem. Note that in the 4th instance we do not report the results for algorithms IP, IPP, or GHRU($r = 1$). The reason is that the memory requirements for building the search space of views and indexes for this instance is simply too large and beyond the limits of the computer capacity.

The first three instances are over a 7-attribute TPC-H dataset. Queries in this dataset are constructed in such a way that the number of aggregated attributes in each of them is a random number between one and four. The last instance is on a 13-attribute TPC-H dataset, where the number of aggregated attributes in each query is a random number between seven and twelve. The number of queries in instances 1,2,3, and 4 is 10, 12, 20, and 41, respectively. As discussed in Section 3, if the value of the storage space is set to $\alpha \times \sum_{q \in Q} size(q) + size(\text{raw-data view})$, then in order for the problem OLAP-VI to be nontrivial, we should have $0 < \alpha < 2$. For the instances presented in Table 1, we set $\alpha = 1$.

In Table 1, for algorithm GHRU we report the results only for the value of parameter r set to 1. (For this algorithm we also solved each instance with r=2, 3, and 4,

---

[6]A detailed report on the entire set of our experimental results can be found in [4].

319

and in each case, we obtained costs identical to those reported for r=1.) Similarly, for algorithm GHRU(ACN) we report the results for the values 1 and 2 of parameter r. (We also solved each instance with r=3 and 4, and in each case the costs we obtained were identical to those for r=2.)

7.2 Observations and Discussion

In every instance we observe that the costs obtained by IP(ACN) and GHRU(ACN) are significantly larger than the corresponding optimal costs, although the associated execution times were consistently low (between 3 to 5 seconds). Furthermore, we note that the cost of the solution obtained by GHRU(ACN) becomes significantly smaller when we increase the value of parameter r from 1 to 2.

We also note that in instances 2, 3, and 4 our IPN model obtains solutions whose costs are relatively close to the optimal values. For the last two instances, our IPNIR models also obtain solutions whose costs are close to the corresponding optimal values. At the same time, the execution times reported for IPN and IPNIR are significantly smaller than those reported for IP and IPP. The two latter models, however, guarantee to obtain the optimal values.

Note that in all of the instances presented in Table 1, the time for building the search space of our IPNIR model is decreasing as we increase the value of parameter p. This supports our intuition that as we increase the value of p, the number of nodes in the graph G_v (related to each view v in the search space of views) decreases. Thus, indexes in IPNIR are built faster for larger values of p.

We observe that in instance 3, although IP cannot solve the problem in our time limit of one hour, IPP provides the optimal solution in only 38.65 seconds. Also, in the first two instances that IP could solve within one hour, the total time needed to solve the instances is much more than the time needed to solve each instance with IPP. We also observed that except for algorithms working with the ACN search space in the input (i.e., IP(ACN) and GHRU(ACN)), in most cases the solving time dominates the space-building time, especially for larger instances.

In the instances that we solved, we observed that the search space of views and indexes for ACN is significantly smaller than the other search spaces of views and indexes. Since in all of the instances that we solved, the costs of the solutions obtained by the IP(ACN) model are significantly larger than the optimal costs, it follows that in these instances algorithm ACN excludes from the search space some of the optimal views and indexes. (Note that the IP(ACN) solution includes the best combination of views and indexes in the ACN search space.) Furthermore, we observe that the number of indexes in the search space of indexes for the IPNIR-S model is mostly smaller than the number of indexes in the search space of other algorithms, except for those algorithms that are applied on the ACN search space.

In instance 4, the total number of views in the lattice, i.e., $|V|$, is 8,191, and the total number of indexes

for all views in V is 6,749,977,113. As discussed in Section 3, the lower bound on the cost value is the number of queries, that is, 41 in this instance. Although in this instance we were not able to solve the problem via either the IP or IPP algorithms (thus we do not know the corresponding optimal value), the cost obtained by the IPN model (45.69) is relatively close to the corresponding lower bound (41). Furthermore, in this instance the search space of IPN with 126 views and 470 indexes is much smaller than the original search space, and we were able to solve this instance using the IPN model in only 36.54 seconds.

We also observed that in those instances that we could solve using GHRU, as we increase the value of r in GHRU from 1 to 2, the amount of time needed to solve the instances decreases significantly. A possible explanation for this behavior is that as the value of r increases, in each step GHRU chooses more indexes, and as a result, the space becomes occupied faster.

Our next observation is related to instance 4. Associated with this instance we constructed two additional instances, by changing the storage-space parameter α to 0.5 and 1.5, respectively, while keeping all the other data unchanged. Our observations on these two instances are consistent with all of our observations above. Further, we observed that the solving time for instances with $\alpha=0.5$ is significantly larger than the solving time for instances with $\alpha=1$ and 1.5 for all of the algorithms except those that are applied on the ACN search space.

When we applied algorithms GHRU and IP on the original search space in instance 4, the computer ran out of memory when building the search spaces of views and indexes; yet we were able to solve this instance using algorithms GHRU(ACN), IP(ACN), and IPN in 4.9, 3.8, and 36.54 seconds, respectively. However, in most of the instances that we solved, the quality of solutions of IP(ACN) and GHRU(ACN) are far from the costs that we got from the IPN model, while the quality of the solutions obtained by IPN was relatively close to the optimal values. Also, note that the quality of the solutions of the IPNIR algorithms does not necessarily increase or decrease as we increase the value of p.

In all of the instances except the first instance, the IPNIR-QS model provides solutions close to the optimum. This supports our intuition that the query nodes and the immediate successors of the root node are more important nodes in graph G_v than the nodes that are only intersections of some queries.

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 search 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 [12] 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 [19, 20], 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, 20] 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

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] Z. Asgharzadeh Talebi, R. Chirkova, Y. Fathi, and M. Stallmann. Exact and inexact methods for selecting views and indexes for OLAP performance improvement. Technical report, NC State University, 2007. Available from `ftp://ftp.ncsu.edu/pub/unity/lockers/ftp/csc_anon/tech/2007/TR-2007-31.pdf`.

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

[6] 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.

[7] 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.

[8] 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.

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

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

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

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

[13] 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.

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

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

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

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

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

[19] 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.

[20] 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.

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

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

[23] 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.

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

[25] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proc. VLDB*, pages 136–145, 97.

inst-ance		no. of indexes	no. of views	cost	Building time (sec.)	Solving time (sec.)	Total time (sec.)
1	IP	31	10062	17.36	0.46	243.00	243.46
1	IPP	31	193	17.36	2.69	0.09	2.78
1	IPN	31	56	109566.00	1.47	0.52	1.99
1	IPNIR p=3	31	56	109566.00	1.46	0.77	2.23
1	IPNIR p=5	31	56	150119.00	1.22	0.33	1.55
1	IPNIR p=8	31	56	508785.00	1.22	0.15	1.37
1	IPNIR Q	31	56	508756.00	1.15	0.51	1.66
1	IPNIR S	31	37	188694.00	1.05	0.57	1.62
1	IPNIR QS	31	56	150119.00	1.32	0.55	1.87
1	GHRU r=1	127	13699	4797034.89	0.08	1736.61	1736.69
1	IP(ACN)	9	16	496290.00	3.59	0.28	3.87
1	GHRU(ACN) r=1	9	16	8994040.56	3.59	<0.01	3.59
1	GHRU(ACN) r=2	9	16	2798807.29	3.59	0.02	3.61
2	IP	40	11226	21.04	0.56	317.96	318.52
2	IPP	40	287	21.04	0.78	0.47	1.25
2	IPN	40	93	21.26	2.17	0.31	2.48
2	IPNIR p=4	40	93	134.72	2.13	0.04	2.17
2	IPNIR p=6	40	93	131.53	2.15	0.17	2.32
2	IPNIR p=10	40	93	375.94	2.14	0.34	2.48
2	IPNIR Q	40	93	376.13	2.12	0.33	2.45
2	IPNIR S	40	59	33.647	2.15	0.02	2.17
2	IPNIR QS	40	59	33.65	2.12	0.02	2.14
2	GHRU r=1	127	13699	6113653.92	0.07	1740.70	1740.77
2	IP(ACN)	14	26	1202.34	3.67	0.31	3.98
2	GHRU(ACN) r=1	14	26	7195552.87	3.67	0.03	3.70
2	GHRU(ACN) r=2	14	26	400421.87	3.67	0.19	3.86
3	IP	60	12742	unknown	0.89	>1 hr	>1 hr
3	IPP	60	651	24.02	0.86	37.79	38.65
3	IPN	60	188	24.11	2.38	0.76	3.14
3	IPNIR p=3	60	184	24.11	2.33	0.61	2.94
3	IPNIR p=6	60	182	27.14	2.30	1.24	3.54
3	IPNIR p=8	60	183	24.78	2.28	0.50	2.78
3	IPNIR p=14	60	183	26.80	2.28	1.62	3.90
3	IPNIR Q	60	184	26.32	2.27	2.00	4.27
3	IPNIR S	60	152	25.84	2.25	1.49	3.74
3	IPNIR QS	60	180	25.66	2.30	1.08	3.38
3	GHRU r=1	127	13699	unknown	0.09	>1hr	>1hr
3	IP(ACN)	27	54	64.43	3.50	0.43	3.93
3	GHRU(ACN) r=1	27	54	3902913.35	3.50	0.03	3.53
3	GHRU(ACN) r=2	27	54	188.03	3.50	0.05	3.55
4	IPN	126	470	45.69	6.91	29.63	36.54
4	IPNIR p=3	126	466	45.77	6.80	17.43	24.23
4	IPNIR p=10	126	446	45.89	5.03	20.42	25.45
4	IPNIR p=20	126	431	46.03	4.56	12.75	17.31
4	IPNIR p=30	126	431	45.87	4.55	10.41	14.96
4	IPNIR Q	126	432	46.15	2.07	51.80	53.87
4	IPNIR S	126	223	47.05	2.87	9.66	12.53
4	IPNIR QS	126	452	46.04	2.88	62.53	65.41
4	IP(ACN)	99	182	55.09	3.84	1.07	4.91
4	GHRU(ACN) r=1	99	182	3898257.15	3.84	0.04	3.88

Table 1: Our experimental results for four representative problem instances.