Selection of Views to Materialize in a Data Warehouse

Himanshu Gupta

Department of Computer Science
Stanford University
hgupta@cs.stanford.edu

Abstract. A data warehouse stores materialized views of data from one or more sources, with the purpose of efficiently implementing decision-support or OLAP queries. One of the most important decisions in designing a data warehouse is the selection of materialized views to be maintained at the warehouse. The goal is to select an appropriate set of views that minimizes total query response time and the cost of maintaining the selected views, given a limited amount of resource, e.g., materialization time, storage space etc.
In this article, we develop a theoretical framework for the general problem of selection of views in a data warehouse. We present competitive polynomial-time heuristics for selection of views to optimize total query response time, for some important special cases of the general data warehouse scenario, viz.: (i) an AND view graph, where each query/view has a unique evaluation, and (ii) an OR view graph, in which any view can be computed from *any one* of its related views, e.g., data cubes. We extend the algorithms to the case when there is a set of indexes associated with each view. Finally, we extend our heuristic to the most general case of AND-OR view graphs.

1 Introduction

A *data warehouse* is a repository of integrated information available for querying and analysis [IK93, Wid95]. Figure 1 illustrates the architecture of a typical warehouse [WGL+96]. The information stored at the warehouse is in the form of views, referred to as *materialized views*, derived from the data in the sources. In order to keep a materialized view consistent with the data at sources, the view has to be *incrementally maintained* [ZGMHW95, GM95]. This maintenance of views incurs what is known as *view maintenance* or *update costs*.

In this paper, we concentrate on the problem of selecting an appropriate set of materialized views, one of the most important design decisions in designing a data warehouse. Given some storage space constraint, the problem is to select a set of derived views to minimize total query response time and the cost of maintaining the selected views. We refer to this problem as the *view-selection problem*.

Related work on this problem has been as follows. [HRU96] presents and analyzes algorithms for selection of views in the special case of "data cubes."

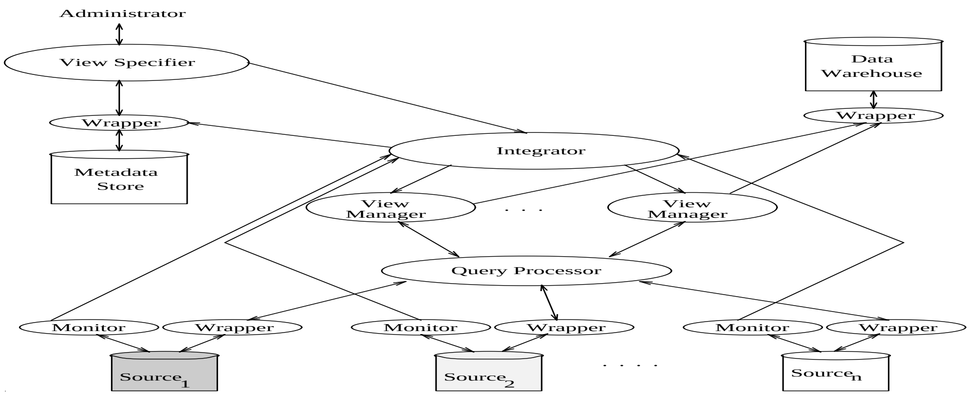

Fig. 1. A typical data warehouse architecture

Gupta et al. in [GHRU96] extend their result to selection of views and indexes in data cubes. Both these works ignore update costs. [RSS96] looks at the problem of augmenting a given set of materialized views with an *additional* set of views that may reduce the total maintenance cost.

The rest of the paper is organized as follows. In the next section, we develop a theoretical framework for the view-selection problem. In the following two sections, we present and analyze heuristics for some special cases. In Section 5, we present an algorithm for the general view-selection problem in a data warehouse. Finally, we conclude in Section 6.

2 View-Selection Problem Formulation

2.1 AND-OR View Graphs

In this subsection, we develop a notion of an AND-OR view graph, which is one of the inputs to the view-selection problem. We start by defining the notions of expression DAGs for queries or views.

Definition 2.1 (**Expression A-DAG**) An *expression A-DAG* (AND-DAG) for a query or a view V is a directed acyclic graph having the base relations as "sinks" (no outgoing edges) and the node V as a "source" (no incoming edges). If a node/view u has outgoing edges to nodes $v_1, v_2, \ldots, v_k$, then *all* of the views $v_1, v_2, \ldots, v_k$ are required to compute u and this dependence is indicated by drawing a semicircle, called an *AND arc*, through the edges $(u, v_1), (u, v_2), \ldots, (u, v_k)$. Such an AND arc has an operator[1] and a cost associated with it, which is the cost incurred during the computation of u from $v_1, v_2, \ldots, v_k$.

The *evaluation cost* of a node u in an expression A-DAG is the sum of the costs associated with each of its descendant's AND arc. □

[1] The operator associated with the AND arc is actually a k-ary function involving operations like join, union, aggregation etc.

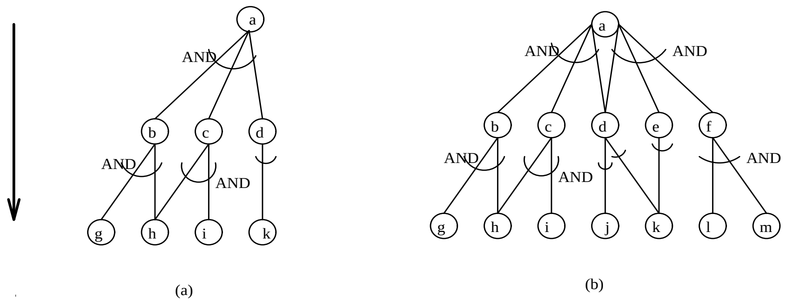

Fig. 2. a) An expression A-DAG, b) An expression AO-DAG

Definition 2.2 (**Expression AO-DAG**) An *expression AO-DAG* for a view or a query V is a directed acyclic graph with V as a source and the base relations as sinks. Each nonsink node has associated with it one or more AND arcs, each binding a *subset* of its outgoing edges. As in the previous definition, each AND arc has an operator and a cost associated with it. More than one AND arc at a node depicts multiple ways of computing that node. □

Definition 2.3 (**AND-OR View Graph**) A graph G is called an *AND-OR view graph* for the views (or queries) $V_1, V_2, \ldots, V_k$ if for each V_i, there is a subgraph G_i in G which is an expression AO-DAG for V_i. Each node u in an AND-OR view graph has the following parameters associated with it: f_u (frequency of the queries on u), S_u (space occupied by u), and g_u (frequency of updates on u). For example, the graph in Figure 2(b) is an AND-OR view graph for any subset of the views a through f. □

Note that in an AND-OR view graph, if a view u can be computed from the views $v, u_1, u_2, \ldots, u_k$ and the view v can be computed from $v_1, v_2, \ldots, v_l$, then u can also be computed from $u_1, u_2, \ldots, u_k, v_1, v_2, \ldots, v_l$.

2.2 Constructing an AND-OR View Graph

Given a set of queries $Q_1, Q_2, \ldots, Q_k$ to be supported at a warehouse, we construct an AND-OR view graph for the queries as follows. We first construct an expression-AO DAG D_i for each query Q_i in the set. An AND-OR view graph G for the set of queries can then be constructed by "merging" the expression AO-DAGs $D_1, D_2, \ldots, D_k$. Each node in the AND-OR view graph G will represent a view that could be selected for materialization, and these are the only views considered for materialization.

For a query Q_i we construct its expression AO-DAG D_i to consist of alternate "useful" ways of evaluating Q_i from the given base relations, in the presence of other queries/views. Roussopoulos in [Rou82] considers exactly this problem. The objective of his analysis is to identify all possible (useful) ways to produce the result of a view, given other view definitions and base relations.

2.3 The View-Selection Problem

Given an AND-OR view graph G and a quantity S (available space), the *view-selection problem* is to select a set of views M, a subset of the nodes in G, that minimizes the sum of total query response time and total maintenance cost, under the constraint that the total space occupied by M is less than S.

More formally, let $Q(u, M)$ denote the cost of answering a query u (also a node of G) using the set M of materialized views in the given view graph G. $Q(u, M)$ is the evaluation cost of the cheapest embedded expression A-DAG for u in G whose sinks belong to the set $M \cup L$, where L is the set of sinks in G. Here, without loss of generality, we have assumed that the sinks in G are always available for computation as they represent the base tables at the source(s). Thus, $Q(u, \phi)$ is the cost of answering the query on u directly from the source(s). Let $U(u, M)$ be the maintenance cost for the view u in the presence of the set of materialized views M and the set of sinks, L.

So, given an AND-OR view graph G and a quantity S, we wish to select a set of views/nodes $M = \{V_1, V_2, \ldots, V_m\}$, that minimizes $\tau(G, M)$, where

$$\tau(G, M) = \sum_{i=1}^{k} f_{Q_i} Q(Q_i, M) + \sum_{i=1}^{m} g_{V_i} U(V_i, M),$$

under the constraint that $\sum_{v \in M} S_v \leq S$.

The view-selection problem is NP-hard even for the special case of an AND-OR graph where each AND arc binds at most one edge, and when the update frequencies are zero. There is a straightforward reduction from `minimum set cover`.

2.4 Benefit of a Set of Selected Views

Let C be an arbitrary set of views in a view graph G. The *benefit* of C with respect to M, an already selected set of views, is denoted by $B(C, M)$ and is defined as $\tau(G, M) - \tau(G, M \cup C)$, where τ is the function defined above. The benefit of C per unit space with respect to M is $B(C, M)/S(C)$, where $S(C)$ is the space occupied by the views in C. Also, $B(C, \phi)$ is called the *absolute benefit* of the set C.

Monotonicity Property The benefit function B is said to satisfy the *monotonicity property* for M with respect to disjoint sets (of views) $O_1, O_2, \ldots, O_m$ if $B(O_1 \cup O_2 \ldots \cup O_m, M) \leq \sum_{i=1}^{i=m} B(O_i, M)$.

The monotonicity property of the benefit function is important for the greedy heuristics to deliver competitive (within a constant factor of optimal) solutions. For a given instance of AND-OR view graph, if the optimal solution O can be partitioned into disjoint subsets of views $O_1, O_2, \ldots, O_m$ such that the benefit function satisfies the monotonicity property w.r.t. $O_1, O_2, \ldots, O_m$, then we guide the greedy heuristic to select, at each stage, an optimal set (of views) of type that includes O_i for all $i \leq m$. Such a greedy heuristic is guaranteed to deliver a solution whose benefit is at least 63% of the optimal benefit, as we show later.

3 AND View Graph

In this section we consider a special case of the view-selection problem in AND-OR view graphs. Here, we assume that each AND arc binds all the outgoing edges from a node. This case depicts the simplied scenario where each view has a unique way of being computed. We call such a graph G an *AND view graph*, where a node can be computed from *all* of its children. As before, each AND arc has an operator and a cost associated with it. An AND view graph for a set of queries is just a "merging" of the expression A-DAGs of the queries.

3.1 Motivation

The general view-selection problem can be approximated by this simplified problem of selecting views in an AND view graph. Given a set of queries supported at the warehouse, instead of contructing an AND-OR view graph as in Section 2.2, we could run a multiple-query optimizer [Sel88, CM82] to generate a global plan, which is essentially an AND view graph for the queries. Such a global plan takes advantage of the common subexpressions among the queries.

3.2 Selection of Views in an AND View Graph

In this subsection, we present heuristics for solving the view-selection problem in AND view graphs without update costs. Later, we extend it to a special case of AND view graphs with update costs. We note here that the view-selection problem in AND view graphs is not known to be NP-complete.

Problem: Given an AND view graph G without updates and a quantity S, find a set of views M that minimizes the quantity $\tau(G, M)$, under the constraint that the total space occupied by the views in M is at most S.

Algorithm 3.1 Greedy Algorithm

Given: G, an AND-OR view graph, and S, the space constraint.
BEGIN
  $M = \phi$; /* M = set of structures selected so far. */
  while $(S(M) < S)$
    Let C be the view which has the maximum benefit per unit space
      with respect to M.
    $M = M \cup C$;
  end while;
  return M;
END.

Greedy Algorithm We present a simple greedy heuristic for selecting views. At each stage, we select a view which has the maximum benefit per unit space at that stage. See Algorithm 3.1. The running time of the greedy algorithm is $O(kn^2)$, where n is the number of nodes in the graph and k is the number of stages used by the algorithm.

Observation 1 *In an AND view graph without updates, the benefit function B satisfies the monotonicity property for any M with respect to arbitrary set of views $O_1, O_2, \ldots, O_m$.*

Theorem 3.1 *For an AND view graph G without updates and a quantity S, the greedy algorithm produces a solution M that uses at most $S + r$ units of space, where r is the size of the largest view in G. Also, the absolute benefit of M is at least $(1 - 1/e)$ times the optimal benefit achievable using as much space as that used by M.*

Proof. It is easy to see that the space used by the greedy algorithm solution, $S(M)$, is at most $S + r$ units. Let $k = S(M)$. Let the optimal solution using k units of space be O and the absolute benefit of O be B.

Consider a stage at which the greedy algorithm has already chosen a set G_l occupying l units of space with "incremental" benefits $a_1, a_2, \ldots, a_l$. The absolute benefit of G_l is thus $\sum_{i=1}^{l} a_i$. Surely the absolute benefit of the set $O \cup G_l$ is at least B. Therefore, the benefit of the set O with respect to G_l, $B(O, G_l)$, is at least $B - \sum_{i=1}^{l} a_i$.

Using Observation 1, it is easy to show by contradiction that there exists a view O_i in O such that $B(O_i, G_l)/|O_i| \geq B(O, G_l)/k$. The benefit per unit space with respect to G_l of the set C selected by the algorithm is at least that of O_i, which is at least $(B - \sum_{i=1}^{l} a_i)/k$. Distributing the benefit of C over each of its unit spaces equally (for the purpose of analysis), we get $a_{l+j} \geq (B - \sum_{i=1}^{l} a_i)/k$, for $0 < j \leq S(C)$. As this is true for each set C selected at any stage, we have the set of equations viz. $B \leq k a_j + \sum_{i=1}^{j-1} a_i$, for $0 < j \leq k$.

Multiplying the j^{th} equation by $(\frac{k-1}{k})^{k-j}$ and adding all the equations, we get $A/B \geq 1 - (\frac{k-1}{k})^k \geq 1 - 1/e$, where $A (= \sum_{i=1}^{k} a_i)$ is the absolute benefit of M. ∎

Greedy-Interchange Algorithm We present another heuristic called the "greedy-interchange" algorithm which starts with the solution produced by the greedy algorithm (Algorithm 3.1) and then improves the solution by interchanging a view already selected with some view not selected.[2] It iteratively performs such interchanging until the solution cannot be improved any further by an interchange. See Algorithm 3.2.

Unfortunately, not much can be proved about the competitiveness of the solution produced by the greedy interchange algorithm except that it is obviously at least as good as the greedy algorithm. Moreover, the running time of the greedy interchange algorithm is unbounded. We believe that the greedy interchange algorithm in practice would perform much better than the greedy algorithm.

[2] When views occupy different amounts of space, more than one view may have to be added/removed.

Algorithm 3.2 <u>**Greedy-Interchange Algorithm**</u>

Given: G, an AND-OR view graph, and S, the space constraint.
Assume that all views occupy the same amount of space.

BEGIN
Run the greedy algorithm and let M be the solution returned.
repeat
Let (C_1, C_2) be a pair of views such that $C_1 \in M$ and the absolute benefit of $(M - C_1) \cup C_2$ is greater than that of M.
$M = (M - C_1) \cup C_2$;
until (no such pair (C_1, C_2) exists);
return M;
END.

3.3 Incorporating Update Costs

Unfortunately, the benefit function may not satisfy the monotonicity property when there are update costs. To see this informally, consider a view C_1 which *helps* in maintaining another view C_2. Hence, the benefit of $C_1 \cup C_2$ might be more than the sum of their benefits individually. However, the benefit function does satisfy the monotonicity property for a special case as shown in the following lemma.

Lemma 1. *In an AND view graph, the benefit function B satisfies the monotonicity property for any M with respect to sets consisting of single views, if the update frequency g_v at any view v is less than its query frequency f_v.*

Proof. It suffices to prove that $B(v, \phi) \geq B(v, M)$ for any view v and a set of views M.

Let A be the set of (not necessarily proper) ancestors of v in the AND view graph G, and let $M_A = A \cap M$. Let D be the set of those ancestors of v which do not have any descendants in the set M.

We have $B(v, \phi) = \sum_{x \in A} f_x(Q(x, \phi) - Q(x, v)) - g_v U(v, \phi)$. Note that, $Q(x, \phi) - Q(x, v) = Q(v, \phi)$ in an AND view graph for any ancestor x of v. Therefore, we get $B(v, \phi) = \sum_{x \in A} f_x Q(v, \phi) - g_v U(v, \phi)$.

For $B(v, M)$, when M has already been selected, v reduces the query costs of only the nodes in D. Therefore, $B(v, M) = \sum_{x \in D} f_x(Q(x, M) - Q(x, M \cup \{v\})) - g_v U(v, M) + \sum_{x \in M_A} g_x(U(x, M) - U(x, M \cup \{v\}))$.

The last term on the right hand side is due to reduction in the update costs of nodes in M_A as a result of the inclusion of v.

As $U(x, M) - U(x, M \cup \{v\}) \leq Q(v, M), \leq Q(v, \phi)$ for any $x \in M_A$, and $(Q(x, M) - Q(x, M \cup \{v\})) = Q(v, M)$ for $x \in D$, we get $B(v, M) \leq \sum_{x \in D} Q(v, M) - g_v U(v, M) + \sum_{x \in M_A} Q(v, \phi)$.

Let M_D be the set of descendants of v in M and let $Q(M_D, \phi) = \sum_{x \in M_D} Q(x, \phi)$. Using $U(v, \phi) - U(v, M) \leq U(M_D, \phi) \leq Q(M_D, \phi)$, and $M_A \cup D \cup \{v\} \subseteq A$, we get $B(v, \phi) - B(v, M) \geq \sum_{x \in D} f_x(Q(v, \phi) - Q(v, M)) + f_v Q(v, \phi) - g_v(Q(M_D, \phi)$.

Now as $Q(v, \phi) - Q(v, M) = Q(M_D, \phi)$, we get $B(v, \phi) - B(v, M) \geq 0$. ∎

Theorem 3.2 *Consider an AND view graph G, where for any view the update frequency is less than its query frequency. For such a graph G, the greedy algorithm produces a solution M whose absolute benefit is at least $(1 - 1/e)$ times the optimal benefit achievable using as much space as that used by M.* ∎

3.4 AND View Graph with Indexes

In this section, we generalize the view-selection problem in an AND view graph by introducing indexes for each node/view. In the presence of indexes the cost of computation depends upon the indexes being used to execute the operation. As indexes are built upon their corresponding views, an index can be materialized only if its corresponding view has already been materialized. Thus, selecting an index without its view does not have any benefit and hence, the benefit function may not satisfy the monotonicity property for arbitrary sets of structures.[3] We assume that if an index is not materialized, then it is never "computed" while answering user queries.

We need to introduce a slightly different cost model for the AND view graphs with indexes. In an AND view graph with indexes, there may be multiple edges from a node u to v, possibly one for each index of v. Instead of associating costs with the arcs, we associate a label (i, t_i) with each edge from u to v. The cost $t_i (i > 0)$[4] can be thought of as the cost incurred in accessing the relation (as many times as required to compute u) at v using its i^{th} index. In addition, we have a k-ary monotonically increasing cost function associated with every arc that binds k edges.

Consider a node u which has k outgoing edges to nodes $v_1, v_2, \ldots, v_k$ and let the k-ary cost function associated with the arc binding all these outgoing edges be f. Then, the cost of computing u from all its children $v_1, v_2, \ldots, v_k$ using their $i_1, i_2, \ldots, i_k^{th}$ indexes respectively is $f(t_{i_1}, t_{i_2}, \ldots, t_{i_k})$, where there is an edge from u to v_j, for $0 < j \leq k$, with a label (i_j, t_{i_j}).

Problem: Given a quantity S and an AND view graph G with indexes. Associated with each edge is a label $(i, t_i), i \geq 0$, and there is a cost function associated with each arc, as described above. Assume that there are no updates.

Find a set of structures M that minimizes the quantity $\tau(G, M)$, under the constraint that the total space occupied by the structures in M is at most S.

Inner-Level Greedy Algorithm The inner-level greedy algorithm works in stages. At each stage, it selects a subset C, which consists either of a view and some of its indexes selected in a greedy manner, or a single index whose view has already been selected in one of the previous stages.

[3] A structure is a view or an index.

[4] When $i = 0, t_0$ is the cost in accessing v without any of its indexes.

Algorithm 3.3 Inner-Level Greedy Algorithm

Given: G, a view graph with indexes, and S, the space constraint.
BEGIN
  $M = \phi$; /* M = Set of structures selected so far */
  while $(S(M) < S)$
    $C = \phi$; /* Best set containing a view and some of its indexes */
    for each view v_i **not** in M
      $IG = \{v_i\}$; /* IG = Set of v_i and some of its indexes selected */
          /* in a greedy manner */
      while $(S(IG) < S)$ /* Construct IG */
        Let I_{ic} be the index of v_i whose benefit per unit space w.r.t.
          $(M \cup IG)$ is maximum.
        $IG = IG \cup I_{ic}$;
      end while;
      if $(B(IG, M)/S(IG) > B(C, M)/|C|)$ **or** $C = \phi$
        $C = IG$;
    end for;
    for each index I_{ij} such that its view $v_i \in M$
      if $B(I_{ij}, M)/S(I_{ij}) > B(C, M)/S(C)$
        $C = \{I_{ij}\}$;
    end for;
    $M = M \cup C$;
  end while;
  return M;
END.

Each stage can be thought of as consisting of two phases. In the first phase, for each view v_i we construct a set IG_i which initially contains only the view. Then, one by one its indexes are added to IG_i in the order of their incremental benefits until the benefit per unit space of IG_i with respect to M, the set of structures selected till this stage, reaches its maximum. That IG_i having the maximum benefit per unit space with respect to M is chosen as C. In the second phase, an index whose benefit per unit space is the maximum with respect to M is selected. The benefit per unit space of the selected index is compared with that of C, and the better one is selected for addition to M. See Algorithm 3.3.

The running time of the inner-level greedy algorithm is $O(k^2m^2)$, where m is the total number of structures in the given AND view graph and k is the maximum number of structures that can fit in S units of space, which in the worst case is S.

Observation 2 *In an AND view graph with indexes and without updates, the benefit function B satisfies the monotonicity property for any M with respect to arbitrary sets of structures $O_1, O_2, \ldots, O_m$, where each O_i consists of a view and some of its indexes.*

Theorem 3.3 *For an AND view graph with indexes and a given quantity S, the inner-level greedy algorithm (Algorithm 3.3) produces a solution M that uses at most $2S$ units of space. Also, the absolute benefit of M is at least $(1-1/e^{0.63}) = 0.467$ of the optimal benefit achievable using as much space as that used by M, assuming that no structure occupies more than S units of space.*

Proof. It is easy to see that $S(M) \leq 2S$. Let $k = |M|$. Let the optimal solution be O, such that $S(O) = k$ and the absolute benefit of O be B.

Consider a stage at which the Inner-level greedy algorithm has already chosen a set G_l occupying l units of space with "incremental" benefits $a_1, a_2, a_3.....a_l$. The absolute benefit of the set $O \cup G_l$ is at least B. Therefore, the benefit of the set O with respect to G_l, $B(O, G_l)$, is at least $B - \sum_{i=1}^{l} a_i$.

If O contains m views, it can be split into m disjoint sets $O_1, O_2, \ldots, O_m$, such that each O_i consists of a view and its indexes in O. By the monotonicity property of B w.r.t. the sets $O_1, \ldots, O_m$, $B(O, G_l) \leq \sum_{i=1}^{m} B(O_i, G_l)$. Now, it is easy to show by contradiction that there exists at least one O_i such that $B(O_i, G_l)/S(O_i) \geq B(O, G_l)/k$. The benefit per unit space of the set C, selected by the Inner-level greedy algorithm at this stage, is at least 0.63 times $B(O_i, G_l)/S(O_i)$. This follows from the result of Theorem 4.1 on the performance guarantee of the simple greedy algorithm (skipping some tedious details here.) Let $k' = 0.63$. Distributing the benefit of C over each of its unit spaces equally (for the purposes of analysis), we get $a_{l+j} \geq k'(B - \sum_{i=1}^{l} a_i)/k$, for $0 \leq j < S(C)$. As this is true for each set C selected at any stage, we have the set of equations viz. $B \leq \frac{k}{k'} a_j + \sum_{i=1}^{j-1} a_i$, for $0 < j \leq k$.

Let $k'' = k/k'$. Multiplying the j^{th} equation by $(\frac{k''-1}{k''})^{k-j}$ and adding all the equations, we get $A/B \geq 1 - (\frac{k''-1}{k''})^{k} \geq 1 - (\frac{k''-1}{k''})^{k''k'} \geq 1 - 1/e^{0.63}$, where $A(= \sum_{i=1}^{k} a_i)$ is the absolute benefit of M. ∎

4 OR View Graph

In this section we consider those AND-OR view graphs in which each AND arc binds exactly one edge. We call such a AND-OR view graph G an *OR view graph*, where a node can be computed from any one of its children.

4.1 Motivation

A specific model of a data warehouse is a data cube. *Data cubes* are databases where a critical value, e.g., `sales`, is organized by several dimensions, for example, sales of automobiles organized by model, color, etc. Queries in such a system are of the usually ask for a breakdown of `sales` by some of the dimensions. Therefore, we can associate an aggregate view, called a *cube*, V_α with each subset α of the dimensions. A view V_α is essentially a result of a "Select α, Sum(`sales`); group by α" SQL query over the base table. An aggregate view V_α can be computed from a view V_β iff $\alpha \subseteq \beta$.

In the data cube, the AND-OR view graph is an OR view graph, as for each view there are zero or more ways to construct it from other views, but each way involves only one other view. Hence, all the results developed in this section for OR view graphs apply to data cubes. As OLAP databases have very few or no updates, we assume that there are no update costs throughout this section.

4.2 View Selection in an OR View Graph

In this subsection, we present algorithms for solving the view-selection problem for OR view graphs without update costs. This generalizes the problem considered by Harinarayan et al. in [HRU96] for selection of cubes in a data cube. We prove that the greedy algorithm (Algorithm 3.1) proposed by them performs with the same performance guarantee even in this setting of an OR view graph. A variant of this problem known as the *K-median* has also been studied in a different context of facility location [CFN77].

Problem: Given an OR view graph G and a quantity S, find a set of views M that minimizes the quantity $\tau(G, M)$, under the constraint that the total space occupied by the views in M is at most S. Assume that there are no updates.

Observation 3 *In an OR view graph without updates, the benefit function B satisfies the monotonicity property for any M with respect to arbitrary sets of views $O_1, O_2, \ldots, O_m$.*

Theorem 4.1 *For an OR view graph G without updates and a given quantity S, the greedy algorithm produces a solution M that uses at most $S + r$ units of space, where r is the size of the largest view in G. Also, the absolute benefit of M is at least $(1 - 1/e)$ times the optimal benefit achievable using as much space as that used by M.* ∎

Recently, Feige in [Fei96] showed that the `minimum set-cover` problem cannot be approximated within a factor of $(1 - o(1)) \ln n$, where n is the number of elements, using a polynomial time algorithm unless $P = NP$. There is a very natural reduction of the `minimum set-cover` problem to our problem of view selection in OR view graphs. The reduction shows that no polynomial time algorithm for the view-selection problem in OR view graphs can guarantee a solution of better than 63% for all inputs unless $P = NP$ [Che96].

Greedy Interchange Algorithm Cornuejols et al. in [CFN77] show for their similar facility location problem through extensive experiments that in most cases the running time of greedy interchange is a little less than 1.5 times the running time of the greedy algorithm, and that it returns a much better solution than that returned by the greedy algorithm.

4.3 OR view graph with Indexes

As in the case of AND view graphs, we generalize the view-selection problem in OR view graphs by introducing indexes for each node/view. In an OR view graph G with indexes, each edge from a node u to v has a label (i, t_i) associated, where $t_i (i > 0)$ is the cost of computing u from v using its i^{th} index and t_0 is the cost of computing u from just v.

Problem: Given a quantity S and an OR view graph G with indexes, find a set of structures M that minimizes the quantity $\tau(G, M)$, under the constraint that the total space occupied by the structures (views and indexes) in M is at most S. Assume that there are no updates.

Observation 4 *In an OR view graph with indexes and without updates, the benefit function B satisfies the monotonicity property for any M with respect to disjoint sets of structures $O_1, O_2, \ldots, O_m$, where each O_i consists of a view and some of its indexes.*

Theorem 4.2 *The Inner-level greedy algorithm produces a solution M that uses at most $2S$ units of space. Also, the absolute benefit of M is at least $(1-1/e^{0.63}) = 0.467$ of the optimal benefit achievable using as much space as that used by M, assuming that no structure occupies more than S units of space.* ∎

5 View Selection in AND-OR View Graphs

In this section, we try to generalize our results developed in the previous sections to the view-selection problem in general AND-OR view graphs. We present here an AO-greedy algorithm that could take exponential time in the worst case, but has a performance guarantee of 63%. We also present a multi-level greedy algorithm which is a generalization of the inner-level greedy algrorithm (Algorithm 3.3). We give a different formulation of the view-selection problem in AND-OR graphs, for the sake of simplifying the description of the algorithm.

Definition 5.1 (**Query-View Graph**) A query-view graph G is a bipartite graph $(Q \cup \zeta, E)$, where Q is the set of queries to be supported at the warehouse and ζ is a subset of the powerset of V, the set of views. An edge (q, σ) is in E iff the query q can be answered using the views in the set σ, and the cost associated with the edge is the cost incurred in answering q using σ. There is also a frequency f_q associated with each query $q \in Q$. We assume that there is a set $\rho \in \zeta$ (the set of base tables) such that $(q, \rho) \in E$ for all $q \in Q$.[5] Note that an arbitrary AND-OR view graph can be converted into an equivalent query-view graph. □

Problem (View Selection in Query-View Graphs): Given a quantity S and a query-view graph $G = (\zeta \cup Q, E)$, select a set of views $M \subseteq V$ that minimizes the total query response time,[6] under the constraint that the total space occupied by the views in M is at most S.

[5] A query-view graph can be looked upon as an OR graph, as a query $q \in Q$ can be computed by any of the *set* of views σ where $(q, \sigma) \in E$.

[6] Though we ignore update costs, it can be incorporated by adding possibly additional nodes in ζ and additional edges in F_ζ (defined later).

5.1 AO-Greedy Algorithm for Query-View Graphs

We define an *intersection graph* F_ζ of ζ as a graph having ζ and D as its set of vertices and edges respectively such that an edge $(\alpha, \beta) \in D$ if and only if the set of views α and β intersect.

The AO-greedy algorithm works in stages as follows. At each stage, the algorithm picks a connected subgraph H of F_ζ whose corresponding set of views V_H (union of the sets of views corresponding to the vertices of H) offers the maximum benefit per unit space at that stage. The set of views V_H is then added to the set of views already selected in previous stages. The algorithm halts when the space occupied by the selected views exceeds S.

We omit the proof of the following theorem due to space constraints.

Observation 5 *An optimal solution O of the view-selection problem in query-view graph G is of the form $O = \cup_{\sigma \in \Gamma} \sigma$, where $\Gamma \subseteq \zeta$.*

Theorem 5.1 *For a query-view graph without updates and a quantity S, the AO-greedy algorithm produces a solution M that uses at most $2S$ units of space. Also, the absolute benefit of M is at least $(1 - 1/e)$ times the optimal benefit achievable using as much space as that used by M.* ∎

For a query-view graph $G = (\zeta \cup Q, E)$ corresponding to an OR view graph, the AO-greedy algorithm behaves exactly as the greedy algorithm (Algorithm 3.1), taking polynomial time for OR view graphs.

5.2 Multi-level Greedy Algorithm

In this section, we generalize the inner-level greedy algorithm (Algorithm 3.3) to multiple inner-levels of greedy selection in query-view graphs. We try to modify the AO-greedy algorithm for query-view graphs in an attempt to improve its running time at the expense of its performance guarantee.

Consider a query-view graph $G = (Q \cup \zeta, E)$ and its intersection graph F_ζ such that there is a view v where $v \in \sigma$ for each node σ in F_ζ.[7] If no such v exists, then run AO-greedy algorithm on G. Let ζ' be the set obtained by removing v from each element of ζ and F'_ζ be its corresponding intersection graph. We select a set of views U whose benefit per unit space is close to that of the optimal.

Let $F_1, F_2, \ldots, F_k$ be the connected components of F'_ζ. We select the set of views U in a greedy manner. Initially the set U contains just v. Then, at each stage, we select a set of views J, corresponding to a subgraph in some component F_i, that has the maximum benefit per unit space. The set of views J is then added to the set U being maintained. We continue adding views to U till the total benefit per unit space of U cannot be further improved.

It is not difficult to show that the benefit per unit space of U at least 63% of the benefit per unit space of V_H, the set of views whose benefit per unit space is

[7] The technique developed here can be easily generalized to the case when F_ζ has $l > 1$ connected components $G_1, G_2, \ldots, G_l$, each satisfying the property that for some v_i, $v_i \in \sigma$ for each vertex σ in G_i.

the maximum among the connected subgraphs in F_ζ. The algorithm continues by interatively picking a new set U and adding it to the set of already selected views M, until the space occupied by M exceeds S.

This algorithm could still take exponential time because of the need to consider all possible subgraphs of F_i. We could apply the above technique recursively for the graphs F_i, selecting a set of views U_i whose benefit is within 63% of the benefit of an optimal set of views in F_i. Applying this technique recursively r times yields the *r-level greedy algorithm*. We omit the proof of the following theorem.

Theorem 5.2 *For a query-view graph G and a given quantity S, the r-level greedy algorithm delivers a solution M that uses at most $2S$ units of space. Also, the benefit of M is at least $1-(1/e)^{0.63^r}$ of the optimal benefit achievable using as much space as that used by M, assuming that no view occupies more than S units of space. The r-level greedy algorithm takes $O((kn)^{2r})$ time, excluding the time taken at the final level. Here, k is the maximum number of views that can fit in S units of space.* ∎

For a given instance one could estimate the value of r such that at the r^{th} level the graphs F_i are small constant-size graphs. The last level would then take only a constant amount of time.

For an OR view graph I with indexes, its equivalent query-view graph $G = (\zeta \cup Q, E)$ is such that each element $\sigma \in \zeta$ consists of a single view and one of its indexes.[8] Hence, at the first stage itself, the graphs obtained consist of nodes representing single indexes. For such a query-view graph, the 1-level inner greedy algorithm behaves exactly the same as the inner-level greedy algorithm (Algorithm 3.3) on OR view graphs with indexes.

6 Conclusions and Future Directions

In this paper, we have developed a theoretical framework for the general problem of selection of views in a data warehouse. We have presented competitive polynomial-time heuristics for some important special cases of the problem that occur in practice. We have presented proofs showing that the algorithms are guaranteed to provide a solution that is within a constant factor of the optimal.

There are still a lot of questions which remain unanswered and need considerable attention. Noteworthy among them are:

1. Are there competitive polynomial-time heuristics for other special cases like AND-OR *trees* or binary AND-OR view trees, even without updates or when optimizing just update costs? Are there heuristics which optimize total query benefit under the constraint of total maintenance time ?
2. Can we prove any negative results about the approximability of the view-selection problem?

[8] Under the assumption that an index is never computed to answer a query.

We believe that the techniques developed in this paper would offer significant insights into the greedy heuristic and the nature of the view-selection problem in a data warehouse. We hope that the view-selection problem would invoke substantial interest in the database theory community.

Acknowledgements

I would like to express my thanks to my advisor, Prof. Jeff Ullman, for his constant encouragement and insightful suggestions.

References

[CFN77] G. Cornuejols, M. L. Fisher, and G. L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithm. *Management Science*, 23(8):789–810, 1977.

[Che96] Chandra Chekuri. Personal Communication, 1996.

[CM82] U. S. Chakravarthy and J. Minker. Processing multiple queries in database systems. *Database Engineering*, 5(3):38–44, September 1982.

[Fei96] U. Feige. A threshold of ln n for approximating set cover. In *Proc. of the* 28^{th} *annual ACM Symp. on the Theory of Comp.*, pages 314–318, 1996.

[GHRU96] H. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman. Index selection in OLAP. Unpublished manuscript. Stanford University, 1996.

[GM95] A. Gupta and I.S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Eng. Bulletin, Special Issue on Materialized Views and Data Warehousing*, 18(2):3–18, 1995.

[HRU96] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing data cubes efficiently. In *ACM SIGMOD Intl. Conf. on Mngt. of Data*, 1996.

[IK93] W.H. Inmon and C. Kelley. *Rdb/VMS: Developing the Data Warehouse.* QED Publishing Group, Boston, Massachussetts, 1993.

[Rou82] N. Roussopoulos. The logical access path schema of a database. *IEEE Transaction in Software Engineering*, SE-8(6):563–573, November 1982.

[RSS96] K. A. Ross, Divesh Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *Proc. of the ACM SIGMOD Int. Conf. on Mngt. of Data*, 1996.

[Sel88] Timos K. Sellis. Multiple query optimization. *ACM Transactions on Database Systems*, 13(1):23–52, March 1988.

[WGL+96] J. Wiener, H. Gupta, W. Labio, Y. Zhuge, H. Garcia-Molina, and J. Widom. A system prototype for warehouse view maintenance. In *Workshop on Materialized Views: Tech. and App.*, 1996.

[Wid95] J. Widom. Research problems in data warehousing. In *Proc. of the* 4^{th} *Intl. Conf. on Info. and Knowledge Mngt.*, pages 25–30, 1995.

[ZGMHW95] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In *Proceedings of the ACM SIGMOD Intl. Conf. on Mngt. of Data*, pages 316–327, 1995.