Selection of Views to Materialize in a Data Warehouse

Himanshu Gupta*[*]
SUNY, Stony Brook

Inderpal Singh Mumick
Kirusa Inc.

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 have developed a theoretical framework for the general problem of selection of views in a data warehouse. We present polynomial-time heuristics for selection of views to optimize total query response time under a disk-space constraint, 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, e.g., when a multiple-query optimizer can be used to general a global evaluation plan for the queries, 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 present proofs showing that the algorithms are guaranteed to provide a solution that is fairly close to (within a constant factor ratio of) the optimal solution. We extend our heuristic to the general AND-OR view graphs. Finally, we address in detail the view-selection problem under the maintenance cost constraint and present provably competitive heuristics.

Keywords: Views, View Selection, Data Warehouse, Materialization.

1 Introduction

Decision support systems have rapidly become a key to gaining competitive advantage for businesses. Business analysts want to run decision support applications to detect business trends by mining the data stored in the information sources. Typically, the information sources maintain historical information, and hence the databases tend to be very large and grow over time. Also, the decision support applications run very complex queries over these information

[*]Contact Author. Dept. of Computer Science, SUNY, Stony Brook. Email: hgupta@cs.sunysb.edu.

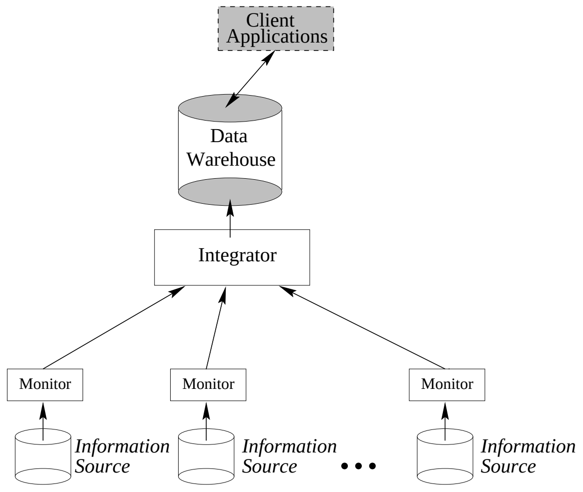


Figure 1: A typical data warehouse architecture

sources. The size of the information source databases and the complexity of queries can cause queries to take unacceptably long to complete. Special purpose query optimization[1, 2, 3] and indexing [4] techniques can reduce query response times to some extent. A commonly used technique to achieve acceptable query response times (in the order of minutes) in such situations is to precompute frequently asked queries and store them in a "data warehouse."

In a typical organization, the information is stored in the form of multiple, independent, and heterogeneous data sources. Functioning as a "data library," a data warehouse makes information readily available for querying and analysis. In essence, a *data warehouse* extracts, integrates, and stores "relevant" information from independent information sources into a central database. The information is stored at the warehouse in *advance* of the queries. In such a system, user queries can be answered using the information stored at the warehouse and need not be translated and shipped to the original source(s) for execution. Also, warehouse data is available for queries even when the original information source(s) are inaccessible due to real-time operations or updates.

Figure 1 illustrates the architecture of a typical data warehouse. The bottom of the figure depicts the multiple *information sources* of interest. Near the top of the figure is the data warehouse, where data that is relevant to the queries to be supported is derived and integrated. Between the sources and the warehouse lie the *source monitors* and the *integrator*. The monitors are responsible for automatically detecting changes in the source data, and reporting them to the integrator. The integrator is responsible for bringing source data into the warehouse, propagating changes in the source relations to the warehouse, and maintaining the tables at the warehouse. Widom in [5] gives a nice overview of the technical issues that arise in the different components of a data warehouse.

The information stored at the warehouse is in the form of derived views of data from the sources. These views stored at the warehouse are often referred to as *materialized views*. Materialized views can speed up the execution of many queries. Any query whose execution plan can be rewritten to use a materialized view is subject to speed-up. For complex queries involving large volumes of data, the speed-up possible using materialized views is dramatic: from hours or days down to seconds or minutes. In fact, materialized views are regarded as one of the primary means for managing performance in a data warehouse [6].

One of the most important issues in a data warehouse design is to select an appropriate set of materialized views to store at the data warehouse. To support a required set of queries at the warehouse, we materialize a set of views that are "closely-related" to the queries. We cannot materialize all possible views, as we are constrained by some resource like disk space, computation time, or maintenance cost. Hence, we need to select an appropriate set of views to materialize under some resource constraint. The *view-selection* problem is defined as selection of a set of views to materialize to minimize the query response time under some resource constraint. In this article, we address the above described view-selection problem in detail and present comprehensive solutions for various special cases and scenarios.

Article Organization. The rest of the article is organized as follows. In the next section, we develop a theoretical framework for the general problem of selecting views to materialize in a data warehouse. In the next two sections, 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 OR view graph, in which any view can be computed from *any one* of its related views, e.g., data cubes, and (ii) an AND view graph, where each query/view has a unique evaluation. For each of the two cases, we extend the algorithms to a more general case when there are index structures associated with the views. In Section 5, we address the view-selection problem in general AND-OR view graphs and present provably competitive algorithms. For all the above described scenarios, we have considered only disk-space as a resource constraint. Then, in Section 6, we look at the view-selection problem under the maintenance-cost constraint. Finally, we end with related work on the view-selection and some concluding remarks. Proofs of all the lemmas appear in Appendix A.

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 1 (**Expression AND-DAG**) An *expression 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. 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 $v_1, v_2, \ldots, v_k$. $\square$

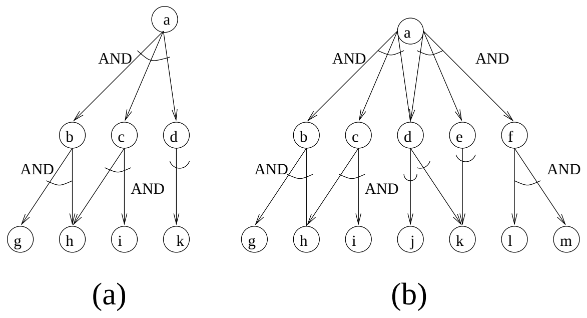


Figure 2: a) An expression AND-DAG, b) An expression ANDOR-DAG

An example of an expression AND-DAG is shown in Figure 2(a). Expression AND-DAGs are more commonly referred to as "expression trees." One inherent drawback of expression AND-DAGs is that they do not depict alternative ways of evaluating a view. The expression ANDOR-DAG, defined next, is a more general notion, which overcomes this shortcoming. An expression ANDOR-DAG may have more than one AND arc at each node, making it an AND/OR expression DAG.

Definition 2 (**Expression ANDOR-DAG**) An *expression ANDOR-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 (called *query-cost*) associated with it. More than one AND arc at a node depicts multiple ways of computing that node. $\square$

Figure 2 shows an example of an expression AND-DAG as well as an expression ANDOR-DAG. In Figure 2 (b), the node a can be computed either from the set of views $\{b, c, d\}$ or $\{d, e, f\}$. The view a can also be computed from the set $\{j, k, f\}$, as d can be computed from j or k and e can be computed from k.

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

Definition 3 (**AND-OR View Graph**) A directed acyclic graph G having the base relations as the sinks 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[2] G_i in G that is an expression ANDOR-DAG for V_i. Each node v in an AND-OR view graph has the following parameters associated with it: space S_v, query-frequency f_v (frequency of the queries on v), update-frequency g_v (frequency of updates on v), and reading-cost R_v (cost incurred in reading the materialized view v). □

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

Definition 4 (**Evaluation Cost**) The *evaluation cost* of an AND-DAG H embedded in an AND-OR view graph G is the sum of the costs associated with the AND arcs in H, plus the sum of the reading costs associated with the sinks/leaves of H. □

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, constructing an AND-OR view graph for the queries involves: (i) constructing an expression ANDOR-DAG D_i for each query q_i, (ii) "merging" the constructed expression ANDOR-DAGs to construct an AND-OR view graph G, and (iii) obtaining the view parameters. The process of constructing an AND-OR view graph for a given set of queries has been addressed comprehensively by Roussopoulos [7]. Below, we briefly discuss each of the three steps involved in construction of an AND-OR view graph. The reader is referred to [7] for further details.

Constructing Query ANDOR-DAGs. Constructing an ANDOR-DAG D_i for a given query q_i involves identifying multiple useful ways of evaluating a query from the given base relations, in the presense of other queries and views. A query/view can be evaluated using multiple expression trees depending on the sequence/order of applying the operators. The problem of building an ANDOR-DAG representing alternate ways of computing a query in the presence of other view definitions and base relations has been addressed by Roussopoulos in [7]. For the special case of AND view graphs, wherein, each AND-arc binds all the outgoing edges from a node, only one evaluation tree for each query is required. Hence, multi-query optimization techniques [8, 9] can be directly used to derive a unique evaluation tree for

[2]An AND-OR view graph H is called a subgraph of an AND-OR view graph G if $V(H) \subseteq V(G)$, $E(H) \subseteq E(G)$, and each edge e_1 in H is bound with the same set of edges through an AND-arc as it is bound through an AND-arc in G. That is, if $e_1, e_2 \in E(G)$, $e_1 \in E(H)$, and e_1 and e_2 are bound by an AND-arc (which may bind other edges too) in G, then $e_2 \in E(H)$, and e_1 and e_2 are bound with the same AND-arc in H. For example, Figure 2 (a) is a subgraph of Figure 2 (b), but Figure 2 (a) without the edge (c, h) is not.

each query, such that the overall performance is optimized. For the special case of OR view graphs, wherein, each AND-arc binds exactly one edge, constructing an AND-OR view graph is equivalent to computing the binary derivability relationship between subexpression views of the queries [10, 11].

Constructing AND-OR View Graph. An AND-OR view graph G for the set of queries can be constructed by integrating or merging the expression ANDOR-DAGs $D_1, D_2, \ldots, D_k$ iteratively. Let G_{i-1} be the AND-OR view graph formed by integrating the ANDOR-DAGs $D_1, \ldots, D_{i-1}$. The process of integration the ANDOR-DAG D_i with G_{i-1} essentially involves: (i) matching nodes in D_i with nodes in G_{i-1} that represent same relational expressions, (ii) identifying whether an unmatched node in D_i can be derived from a set of nodes in G_{i-1} or vice-versa, and (iii) embedding the derived relationships identified in (ii). Each node in the final AND-OR view graph G_k will represent a view that could be selected for materialization.

Computing View Parameters. The query frequencies (f_v) are computed based on the query workload of the data warehouse. A simple method for computing query frequencies is based on the established query usage patterns, if the system has been operational long enough to establish valid patterns. Update frequencies (g_v) of a view is the sum of the update frequencies of all the base relations used for derivation of the view. Here, we only consider direct updates on base relations and assume that the updates on base relations are independent. The space parameter S_v for a view v can be determined by computing the expected number of tuples in the view.

The above discussion gives a general idea of constructing an AND-OR view graph for a given set of queries, based on previous works [7, 8, 10, 11]. In the rest of the article, we assume that we are given a view graph and we focus our attention on selecting an optimal set of views for materialization from a given view graph.

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 total query response time, under the constraint that the total space occupied by M is less than S. In Section 6, we will address the problem of selection of views under a maintenance-cost constraint.

More formally, let $Q(v, M)$ denote the cost of answering a query v (also a node of G) using the set M of materialized views in the given view graph G, and $UC(v, M)$[3] be the maintenance cost (due to updates to base tables) for the view v in the presence of the set of materialized views M. We will always assume that the set of sinks L is also available for querying and

[3]The function symbol UC denotes *update cost.*

maintenance purposes. Then, given an AND-OR view graph G for queries $q_1, \ldots, q_k$ and a quantity S, the view-selection problem is to select a set of views/nodes $M = \{V_1, V_2, \ldots, V_m\}$, that minimizes $\tau(G, M)$, where

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

under the constraint that $\sum_{v \in M} S_v \leq S$. Recall that S_v is the space occupied by the view v.

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

Computing Q(v,M). The cost of answering a query v in presence of a set of (materialized) views M, $Q(v, M)$, in an AND-OR view graph G is actually the evaluation cost of the cheapest AND-DAG H_v for v, such that H_v is a subgraph of G and the sinks of H_v belong to the set $M \cup L$, where L is the set of sinks in G. Again, we have assumed that L, the set of sinks in G, is available for computation as it represents the set of base tables at the source(s). Thus, the value $Q(v, \phi)$ is the cost of answering a query on v directly from the source(s). For special AND-OR view graphs (called OR view graphs), wherein each AND arc binds exactly one edge, the query cost $Q(v, M)$ is the minimum query-length of a path from v to some $u \in (M \cup L)$, where the *query-length* of a path from v to u is defined as R_u, the reading cost of u, plus the sum of the query-costs associated with the edges on the path.

In this article, we have ignored maintenance costs except in Section 4.3 and Section 6. Section 4.3 incorporates update costs in the objective function τ, while Section 6. considers the view-selection problem under the maintenance-cost constraint. As the results in Section 4.3 are independent of maintenance cost models, we defer the discussion on maintenance cost models and computation of $UC(v, M)$ until Section 6.

2.4 Benefit of a Set of Selected Views

In this subsection, we define the notion of a "benefit" function, which is central to the development of algorithms presented in this article. In the following two sections, we will present approximation algorithms for some special cases of the general view-selection problem.

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.

2.4.1 Monotonicity Property

The benefit function B is said to satisfy the *monotonicity property* for M with respect to sets (of views) $O_1, O_2, \ldots, O_m$ if $B(O_1 \cup O_2 \ldots \cup O_m, M) \leq \sum_{i=1}^{i=m} B(O_i, M)$.[4]

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. In the following two sections, we discuss various special cases of AND-OR view graphs (OR view graphs and AND view graphs) wherein the benefit function satisfies the monotonicity property.

3 OR View Graphs

In this section, we consider a special case of the view-selection problem for AND-OR view graphs. We restrict our attention to those AND-OR view graphs in which each AND arc binds exactly one edge. For such restricted AND-OR view graphs, we can remove AND arcs altogether, and associate the costs and the operators with the corresponding edges in the graph. 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.

3.1 Motivation

The OR view graphs arise in many useful practical applications when computation of a view depends on only one other view. A simple application is when all the views and queries involved are aggregate queries over the base data. Data cubes is another example of OR view graphs.

Data Cubes. In a data cube users can view the data as multidimensional data. *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, day of sale, place of sale, age of purchaser and so on. The metric of interest is called the *measure attribute*, which is `sales` in the above example. Queries in such a system are of the OLAP (On line Analytic Processing) type, usually asking 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

[4]Considering $m = 2$ is sufficient, but we state it for general m so that its application is direct.

V_α is essentially a result of a "Select α, Sum(`sales`); group by α" SQL query over the base table. Hence, an aggregate view V_α can be computed from a view V_β iff $\alpha \subseteq \beta$.

In a 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. Data cubes being a special case of OR view graphs, all the results developed in this section apply to data cubes. As OLAP databases have very few or no updates at the base table, we assume that there are no maintenance costs at the materialized views throughout this section.

3.2 Selection of Views in an OR View Graph

In this subsection, we present heuristics for solving the view-selection problem in OR view graphs without maintenance costs.

Problem: Given an OR 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.

3.2.1 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. The greedy heuristic is presented below as Algorithm 1.

Algorithm 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 that has the maximum benefit per unit space with respect to $M$.
    $M = M \cup C$;
  end while;
  return M;
  END.                                                                    ◇
```

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 OR 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 1 *For an OR 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 $a_1, a_2, \ldots, a_l$. Incremental benefit a_i is defined as the increase in benefit of M, when the i^{th} unit of space is added to M. Thus, the absolute benefit of G_l is $\sum_{i=1}^{l} a_i$. 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 $B - \sum_{i=1}^{l} a_i$.

Let $O = \{O_1, O_2, \ldots, O_m\}$. By the monotonicity property of the benefit function for the views O_i's, $B(O, G_l) \leq \sum_{i=1}^{m} B(O_i, G_l)$. Now, we show by contradiction that there exists a view O_i in O such that $B(O_i, G_l)/|O_i| \geq B(O, G_l)/k$. Let us assume that there is no such view O_i in O. Then, $B(O_j, G_l) < (B(O, G_l)/k) * |O_j|$ for every view $O_j \in O$. Thus, $\sum_{O_j \in O} B(O_j, G_l) < (B(O, G_l)/k) * \sum_{O_j \in O} |O_i| = B(O, G_l)$, which violates the monotonicity property of the benefit function for the views $O_j \in O$. Therefore, there exists a view O_i in O such that $B(O_i, G_l)/|O_i| \geq B(O, G_l)/k \geq (B - \sum_{i=1}^{l} a_i)/k$.

The benefit per unit space with respect to G_l of the view C selected by the algorithm is at least that of O_i, which is at least $B(O, G_l)/k = (B - \sum_{i=1}^{l} a_i)/k$, as shown above. 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 the above analysis is true for each view C selected at any stage, we have

$$B \leq k a_j + \sum_{i=1}^{j-1} a_i \qquad \text{for} \quad 0 < j \leq k.$$

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. ∎

If we are required not to exceed the space constraint S, then we consider for selection only those views that have a space of less than S, and pick the better of the following two solutions: (i) greedy solution of Algorithm 1 minus the last view added, or (ii) the solution consisting of just one view that has the highest query benefit. It can be easily shown that the better of the above two solutions will have a query benefit of at least $(1 - 1/e)/2$ times the optimal benefit achievable.

Feige in [12] 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 $P = NP$.

3.3 OR View Graph with Indexes

In this section, we generalize the view-selection problem in an OR view graph by introducing indexes for each node/view. As in the original OR view graph, a node can be computed from any one of its children, but in the presence of indexes the cost of computation depends upon the index 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 the benefit of an index actually increases with the materialization of its view. Hence, the benefit function may not satisfy the monotonicity property for arbitrary sets of views and indexes. We use the term *structure* to denote a view or an index. We assume that if an index is not materialized, then it is never "computed" while answering user queries. In our earlier work [11], we determine the search space of indexes to be considered for materialization.

In most commercial systems today, the views that are to be precomputed are selected first, followed by the selection of the appropriate indexes on them. A trial-and-error approach is used to divide the space available between the summary tables and the indexes. This two-step process can perform very poorly. Since both views and indexes consume the same resource - space - their selection should be done together for the most efficient use of resources. In this section, we present a family of algorithms of increasing time complexities, and prove strong performance bounds for them.

We need to introduce a slightly different cost model for the OR view graphs with indexes. In an OR 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 a cost with the edges, we associate a label (i, t_i) with each edge from u to v. The label $t_i (i > 0)$ can be thought of as the cost incurred in computing u from v using its i^{th} index. When $i = 0, t_0$ is the cost in computing u from v without any of its indexes.

<u>Problem</u>: Given a quantity S and an OR view graph G with indexes. Associated with each edge is a label $(i, t_i), i \geq 0$ 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.

3.3.1 Inner-Level Greedy Algorithm

The inner-level greedy algorithm works in stages. At each stage, it selects a subset C, which consists of either 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.

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 2.

Algorithm 2 **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 ($S(M) < S$)
    $C = \phi$;                                 /* Set of structures to be selected */
    for each view $v_i$ 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.                                                                      $\diamond$
```

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 OR 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 OR 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 2 *For an OR view graph with indexes and a given quantity S, the inner-level greedy algorithm (Algorithm 2) 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 $a_1, a_2, a_3 \ldots\ldots a_l$. 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 V_i and its indexes in O. By the monotonicity property of the benefit function 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 $B(O_i, G_l)/S(O_i) \geq B(O, G_l)/k$ (else $B(O, G_l) > \sum_{i=1}^{m} B(O_i, G_l)$).

In this paragraph, we show that 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 $B(O_i, G_l)/S(O_i)$. Without loss of generality, we assume that the view V_i in O_i has not been selected.[5] Consider the greedy solution G of the indexes of V_i of space $S(O_i) - S(V_i)$, when $G_l \cup V_i$ has already been selected. The benefit of G is at least 63% of the optimal, from the result of Theorem 1. Hence,

$$B(G, G_l \cup \{V_i\}) \geq 0.63 B(O_i - \{V_i\}, G_l \cup \{V_i\}),$$

as $O_i - \{V_i\}$ is also a solution (possibly non-optimal). Now,

$$\begin{aligned} B(G \cup \{V_i\}, G_l) &= B(V_i, G_l) + B(G, G_l \cup \{V_i\}) \\ &\geq B(V_i, G_l) + 0.63 B(O_i - \{V_i\}, G_l \cup \{V_i\}) \\ &\geq 0.63 B(O_i, G_l) \end{aligned}$$

[5]If the view $V_i \in O_i$ has already been selected, then C is at least as good as O_i's best index not yet selected. In that case, the benefit per unit space of C is obviously at least $B(O_i, G_l)/S(O_i)$.

As the inner-level greedy algorithm, while selecting indexes greedily, stops when the benefit per unit space of C becomes maximum, the benefit per unit space of C is at least that of $G \cup \{V_i\}$. Therefore, $B(C, G_l)/|C| \geq B(G \cup \{V_i\}, G_l)/S(O_i) \geq 0.63B(O_i, G_l)/S(O_i)$.

Since, O_i is such that $B(O_i, G_l)/S(O_i) \geq B(O, G_l)/k$, we get $B(C, G_l)/|C| \geq 0.63B(O, G_l)/k \geq 0.63(B - \sum_{i=1}^{l} a_i)/k$. Now, 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 the above analysis is true for each set C selected at any stage, we have

$$B \leq \frac{k}{k'} a_j + \sum_{i=1}^{j-1} a_i \qquad \text{for } \ 0 < j \leq k.$$

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 AND View Graph

In this section, we consider another 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 is 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 AND-DAGs of the queries. We omit the proofs of the theorems in this section as they are similar to that of the corresponding theorems in Section 3.

4.1 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 constructing an AND-OR view graph as in Section 2.2, we could run a multiple query optimizer [8, 13] 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. Figure 3 shows an example of an AND view graph, a global plan, for the queries $R \bowtie S \bowtie T$ and $R \bowtie S \bowtie U$.

4.2 View Selection in an AND View Graph

In this subsection, we show that the greedy algorithm (Algorithm 1) can also be applied to solve the view-selection problem in AND view graphs without maintenance costs. In the later

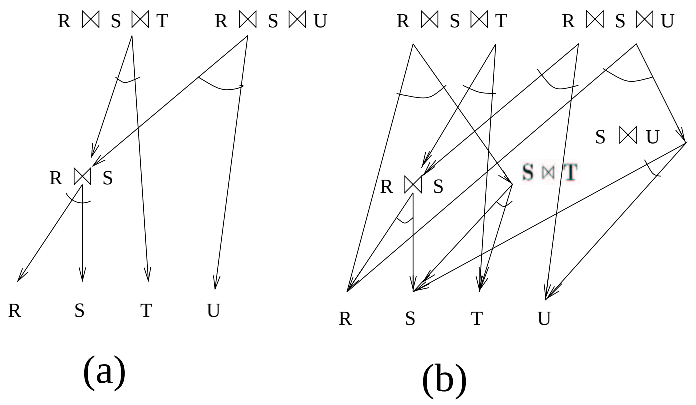


Figure 3: (a) An AND view graph. (b) An AND-OR view graph

subsection, we extend it to a special case of AND view graphs with maintenance costs.

Problem: Given an AND 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 AND 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 3 *For an AND 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.* ∎

4.3 Incorporating Maintenance Costs in τ

Unfortunately, the benefit function may not satisfy the monotonicity property when maintenance costs are included in the objective function (τ). To illustrate the nonmonotonicity of the benefit function, consider a view C_1 that *helps* in maintaining another view C_2. Then, the benefit of $C_1 \cup C_2$ might be more than the sum of their benefits individually.

We show that when the update frequency at any node/view is less than its query frequency, i.e., when the number of times a view is updated (number of batch updates) is less than than the number of times it is queried, the benefit function does satisfy the monotonicity property in AND view graphs. Thus, for this special case of AND view graph, the solution returned by the greedy algorithm is guaranteed to have a benefit of at least 63% of the optimum benefit.

Lemma 1 *In an AND view graph,*[6] *$B(v, \phi) \geq B(v, M)$ for any view v and a set of views M, if the update frequency g_x at any view x is less than its query frequency f_x.* ∎

[6]The claim doesn't hold for OR view graphs.

Lemma 2 *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.* ∎

Theorem 4 *Consider an AND view graph G with updates, 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.* ∎

4.4 AND View Graph With Indexes

As in the case of OR view graphs, we generalize the view-selection problem in AND view graphs by introducing indexes for each node/view. As in the original AND view graph, a node can be computed from all of its children, but in the presence of indexes the cost of computation depends upon the indexes being used to execute the operation.

We need to introduce a slightly different cost model for the AND view graphs with indexes. In an AND view graph with indexes, 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 $(i > 0)$[7] 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 that 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, 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 AND 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 $O_1, \ldots, O_m$, where each O_i consists of a view and some of its indexes.*

Theorem 5 *For an AND graph, 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.* ∎

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

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. Unfortunately, we couldn't devise a polynomial time algorithm for the general AND-OR view graphs that delivers a competitive solution. Instead, we present here an AO-greedy algorithm (a modification of the greedy heuristic) that could take exponential time in the worst case, but has a performance guarantee of 63%. We show that the AO-greedy algorithm developed here runs in polynomial time when the view graph is an OR view graph. We also present a multi-level greedy algorithm which is a generalization of the inner-level greedy algorithm (Algorithm 2). The proofs of theorems in this section are somewhat similar to previous proofs and can also be found in [14].

We give a different formulation of the view-selection problem in AND-OR graphs, for the sake of simplifying the presentation. First, we define the notion of query-view graphs.

Definition 5 (**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 power set 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.[8] 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 $q \in Q$. $\square$

Note that any given AND-OR view graph can be converted into an equivalent query-view graph. The size of the resulting query-view graph is equal to the number of sets of views that help answer a query, and may be exponential in the size of the original AND-OR view graph. We now formulate the view-selection problem in a 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,[9] under the constraint that the total space occupied by the views in M is at most S.

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 sets of views α and β intersect.

[8] 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 $(q, \sigma) \in E$.

[9] Though we ignore maintenance costs, it can be incorporated by adding additional nodes in ζ.

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 M, the set of views already selected in previous stages. The algorithm halts and returns M when the space occupied by M exceeds S.

The time complexity of the AO-greedy algorithm is exponential in the number of *edges* in the intersection graph F_ζ. Thus, AO-greedy algorithm may be practical for query-view graphs that have very sparse intersection graphs. To improve the running time, after the selection at each stage, we can change the set ζ by removing the selected views V_H from each element (a set of views) in ζ. Graph F_ζ, for the next stage, is then reconstructed from the new ζ.

Lemma 3 *An optimal solution O of the view-selection problem in query-view graph $G = (\zeta \cup Q, E)$ can be partitioned into sets of views $O_1, O_2, \ldots, O_m$, such that each O_i corresponds to a connected subgraph in F_ζ, as defined above, and $B(O, M) \leq \sum_{i=1}^{m} B(O_i, M)$.* ■

Theorem 6 *For a query-view graph 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.* ■

The equivalent query-view graph $G = (\zeta \cup Q, E)$ of an OR view graph is such that each element $\sigma \in \zeta$ consists of exactly one view and hence F_ζ has zero edges. For such a graph G, the AO-greedy algorithm behaves exactly as the greedy algorithm (Algorithm 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 2) to multiple 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 the intersection graph F_ζ of ζ. Let F_ζ have $l > 1$ connected components and the let $G_1, G_2, \ldots, G_l$ where $G_i = (Q \cup \zeta_i, E_i)$ be the corresponding query-view subgraphs of G. The multi-level inner greedy algorithm works in stages. At each stage, it searches for a set of views W_i in each G_i, such that the benefit per unit space of W_i is maximum. Each set W_i is computed by using the recursive function `InnerGreedy` on ζ_i. Among all W_i's, the set W_i that has the maximum benefit per unit space is added to the solution M being maintained. The solution M is returned when the total space constraint has been consumed.

The recursive function `InnerGreedy` works as follows. Let the input be the set of nodes Γ. Let us assume that there is a view v where $v \in \sigma$ for each node σ in Γ. If no such v exists, then the InnerGreedy function does an exhaustive search (or run the AO-greedy algorithm) and return a set of views that has the optimal benefit per unit space. If v exists, let $\Gamma_1, \Gamma_2, \ldots, \Gamma_m$ be the sets corresponding to the connected components of the resulting intersection graph. The set of views U, that has to be returned by the InnerGreedy function, is selected in the following greedy manner. Initially the set U contains only v. Then, at each stage, we search recursively in each Γ_i for a set of views J_i that has the maximum benefit per unit space. The set J_i that has the maximum benefit per unit space is added to the set U being maintained. We continue adding views to U until the total benefit per unit space of U cannot be further improved. At that point, the set U is returned.

The multi-level (r-level) greedy algorithm and the InnerGreedy function is shown below as Algorithm 3.

Algorithm 3 <u>**Multi-level (r-level) Greedy Algorithm**</u>

```
Given: A query-view graph $G = (Q \cup \zeta, E)$ and the space constraint $S$.
BEGIN
  $M = \phi$;                              /* $M$ = set of structures selected so far. */
  while ($S(M) < S$)
    Let $G_1, G_2, \ldots, G_l$ be the connected components of the intersection graph $F_\zeta$ and
      let $\zeta_1, \ldots, \zeta_m$ be the corresponding subsets of $\zeta$.
    For each $i \leq m$, $W_i$ = InnerGreedy($r, \zeta_i, M$);
    Let $W$ be the $W_i$ that has the maximum benefit per unit space;
    $M = M \cup W$;
    Reduce $\zeta$ by removing the views in $W$ from each of its elements;
  end while;
  return M;
END.
Function InnerGreedy($r, \Gamma, M$)       /* Returns a set of views $U$ that has the best
                  benefit per unit space. The main inputs $G$ and $S$ are globally defined. */
BEGIN
  If $r = 0$, pick $U$ by doing exhaustive search;
  Let $F_\Gamma$ be the intersection graph of $\Gamma$.
  Let $v$ be such that for all $\sigma \in \Gamma$, $v \in \sigma$.
    If no such $v$ exists, pick $U$ by doing exhaustive search.
  Let $\Gamma_1, \ldots, \Gamma_m$ be the corresponding subsets of $\Gamma$
    obtained after removing the view $v$ from each element.
  $P = 0$;   $U = \{v\}$;
```

```
    **while** $(S(U) < S)$
        For each $i$, let $J_i = \text{InnerGreedy}(r-1, \Gamma_i, (M \cup U))$;
        Let $J$ be the $J_i$ with the maximum benefit per unit space.
        if $(B(U \cup J, M)/S(U \cup J) \leq B(U, M)/B(U))$
            **return** $U$;
        $U = U \cup J$;
    **end while;**
    **return** $U$.
**END.**
```
$\diamond$

Lemma 4 *The InnerGreedy function with the first parameter value equal to r delivers a solution U whose benefit per unit space is at least $g(r)$ of the optimal benefit per unit space achievable. The function $g(r)$ is defined recursively as $g(r) = 1 - 1/e^{g(r-1)}$, and $g(0) = 1$.* ∎

Theorem 7 *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 $g(r+1)$ times the optimal benefit achievable using as much space as that used by M, assuming that no view occupies more than 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. The r-level greedy algorithm takes $O((kn)^{2r})$ time, excluding the time taken at the final level, where k is the maximum number of views that can fit in S units of space. Also, the values of the function $g(r)$ for increasing values of r are 1, 0.63, 0.46, 0.37, 0.31, 0.26, 0.23 and so on.

The equivalent query-view graph $G = (\zeta \cup Q, E)$ of an OR view graph with indexes is such that each element $\sigma \in \zeta$ consists of a single view and one of its indexes. For such a query-view graph G, the 1-level greedy algorithm behaves exactly the same as the inner-level greedy algorithm (Algorithm 2) on OR view graphs with indexes. The 2-level greedy algorithm is very well suited for the case of OR graph with index, where even the indexes are indexed. So, r-level greedy algorithm is well suited for OR or AND view graphs with r-level indexing schemes.

6 View-Selection Under a Maintenance Cost Constraint

In the previous sections, we have looked at the view-selection problem under the disk-space constraint.[10] Now, we consider the view-selection problem under the constraint that the

[10]Section 4.3 only incorporated maintenance costs in the objective function (τ).

selected set of views incur less than a given amount of total maintenance time. Hereafter, we will refer to this problem as the *maintenance-cost view-selection* problem.

In practice, the real constraining factor that prevents us from materializing everything at the warehouse is the maintenance time incurred in keeping the materialized views up to date at the warehouse. Usually, changes to the source data are queued and propagated periodically to the warehouse views in a large batch update transaction. The update transaction is usually done overnight, so that the warehouse is available for querying and analysis during the day time. Hence, there is a constraint on the time that can be allotted to the maintenance of materialized views. Thus, the maintenance-cost view-selection problem is of much practical importance.

Section Organization. We start with a formal definition of the view-selection problem under the maintenance cost constraint. Section 6.2 shows how to extend view graphs to incorporate maintenance costs. We present our approximation algorithm for the maintenance-cost view-selection problem in Section 6.3. We end the section with experimental results that indicate that the proposed greedy heuristic almost always returns an optimal solution for OR view graphs and runs must faster than the A^* heuristic.

6.1 The Maintenance-Cost View-Selection Problem

In this section, we present a formal definition of the maintenance-cost view-selection problem, which is to select a set of views in order to minimize the total query response time under a given maintenance-cost constraint.

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

Let $Q(v, M)$ denote the cost of answering a query v (also a node of G) in the presence of a set M of materialized views. As defined in the previous section, let $UC(v, M)$ is the cost of maintaining a materialized view v in presence of a set M of materialized views. The maintenance-cost view-selection problem is formally formulated as follows. Given an AND-OR view graph G and a quantity S, the maintenance-cost view-selection problem is to select a set of views/nodes M, that minimizes the *objective function* $\tau(G, M)$, where

$$\tau(G, M) = \sum_{v \in V(G)} f_v Q(v, M),$$

under the constraint that $U(M) \leq S$, where $U(M)$, the *total maintenance time*, is defined as

$$U(M) = \sum_{v \in M} g_v UC(v, M).$$

The view-selection problem under a disk-space constraint can be easily shown to be `NP`-hard, as there is a straightforward reduction [15] from the `minimum set cover` problem. The view-selection problem under a disk-space constraint is a special case of the maintenance-cost view-selection problem, when the maintenance cost of each view remains constant. Thus, the maintenance-cost view-selection problem, which is a more general problem, is trivially `NP`-hard.

Space vs. Maintenance-Cost constraint. The main difficultly in the maintenance-cost view-selection problem arises from the fact that the maintenance cost of a view v depends on the set of other materialized views, whereas in the space-constraint view-selection problem the space associated with a view remains constant. Thus, even when the query-benefit function satisfies the monotonicity property (Section 2.4.1), the query-benefit *per unit of maintenance-cost* of a view can actually increase, since the maintenance cost of a view can decrease with the selection of other views. This nonmonotonic behavior of the query-benefit *per unit constraint* function can cause the query-benefit per unit maintenance-cost of two "dependent" views to be sometimes much greater than the query-benefit per unit maintenance-cost of either of the individual views. Thus, the simple greedy approach of selecting the best individual view at each stage can perform arbitrarily bad. Example 2 in Section 6.3 illustrates the above described nonmonotonic behavior of the query-benefit per unit maintenance-cost function and shows that the simple greedy approach could deliver an arbitrarily bad solution.

6.2 Incorporating Maintenance Costs in View Graphs

We need to incorporate maintenance costs in the definition of AND-OR view graphs, so that the input to the view-selection maintenance-cost problem is complete. So, with each AND-OR graph defined there is a maintenance-cost function UC associated with it. The function UC is such that for a view v and a set of views M, $UC(v, M)$ gives the cost of maintaining v in presence of the set M of materialized views. We assume that the set L of base relations in G is always available. In this article, we do not discuss various maintenance cost models possible for an AND-OR view graph, and hence we assume that the function UC is given as part of an input with the AND-OR graph.

Maintenance Costs in OR View Graphs. In case of OR view graphs, instead of the maintenance cost function UC for the graph, there is a maintenance-cost value associated with each edge (u, v), which is the maintenance cost incurred in maintaining u using the materialized view v. Figure 4 shows an example of an OR view graph $\mathcal{G}$ with the associated maintenance-costs. In OR view graphs with maintenance costs, let us define *maintenance-length* of a path in the OR graph as the sum of the maintenance-costs associated with the edges

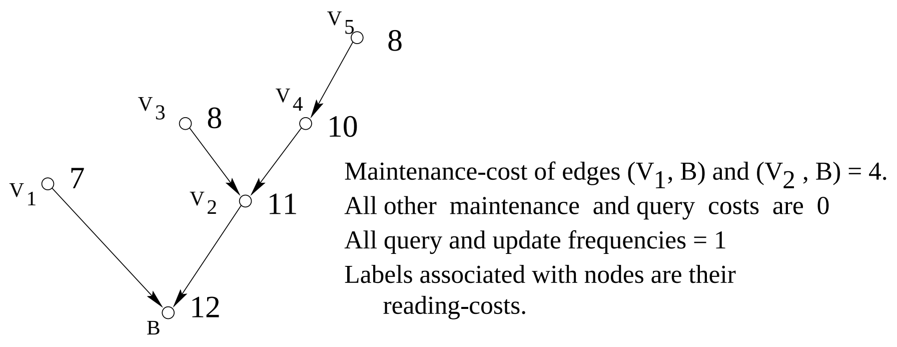

V5 8
V4 10
V3 8
V1 7
V2 11
B 12
Maintenance-cost of edges (V1, B) and (V2 , B) = 4.
All other maintenance and query costs are 0
All query and update frequencies = 1
Labels associated with nodes are their reading-costs.

Figure 4: $\mathcal{G}$: An OR view graph

on the path.[11] Then, the maintenance cost $UC(v, M)$ is equal to the minimum maintenance-length of some path from v to another node u in $(M \cup L)$. The above characterization of $UC(v, M)$ in OR view graphs is without any loss of generality of a maintenance-cost model, because in OR view graphs a view u uses at most one view to help maintain itself.

In the following example, we illustrate how to compute $Q(v, M)$ and $UC(v, M)$ on OR view graphs.

EXAMPLE 1 Consider the OR view graph $\mathcal{G}$ of Figure 4, where B is the base table and $V_1, \ldots, V_5$ are the views. The maintenance-costs and query-costs associated with each edge is zero, except for the maintenance-cost of 4 associated with the edges (V_1, B) and (V_2, B). Also, all query and update frequencies are uniformly 1. The label associated with each of the nodes in $\mathcal{G}$ is the reading-cost of the node. Also, the set of sinks $L = \{B\}$.

In the OR view graph $\mathcal{G}$, $Q(V_i, \phi) = 12$ for all $i \leq 5$, because as the query-costs are all zero, the minimum query-length of a path from V_i to B is just the reading-cost of B. Note that $Q(B, \phi) = 12$. Also, as the minimum maintenance-length of a path from a view V_i to B is 4, $UC(V_i, \phi) = 4$ for all $i \leq 5$. $\square$

6.3 Inverted-Tree Greedy Algorithm

In this section, we present a competitive greedy algorithm called the *Inverted-Tree Greedy Algorithm* which delivers a near-optimal solution for the maintenance-cost view-selection problem in OR view graphs.

One of the key notions required in designing a greedy algorithm for selection of views is the notion of the "most beneficial" view. In the greedy heuristics proposed in Section 2 for selection of views to materialize under a space constraint, views are selected in order of their

[11]Note that the maintenance-length doesn't include the reading cost of the destination as in the query-length of a path.

"query benefits" per unit space consumed. We now define a similar notion of benefit for the maintenance-cost view-selection problem addressed in this section.

Most Beneficial View. Consider an OR view graph G. At a stage, when a set of views M has already been selected for materialization, the *query benefit* $B(C, M)$ associated with a set of views C with respect to M is defined as $\tau(G, M) - \tau(G, M \cup C)$. We define the *effective maintenance-cost* $EU(C, M)$ of C with respect to M as $U(M \cup C) - U(M)$.[12] Based on these two notions, we define the view that has the most query-benefit per unit effective maintenance-cost with respect to M as the *most beneficial view* for greedy selection at the stage when the set M has already been selected for materialization.

We illustrate through an example that a *simple greedy* algorithm, that at each stage selects the most beneficial view, as defined above, could deliver an arbitrarily bad solution.

<u>EXAMPLE</u> 2 Consider the OR view graph $\mathcal{G}$ shown in Figure 4. We assume that the base relation B is materialized and we consider the case when the maintenance-cost constraint is 4 units.

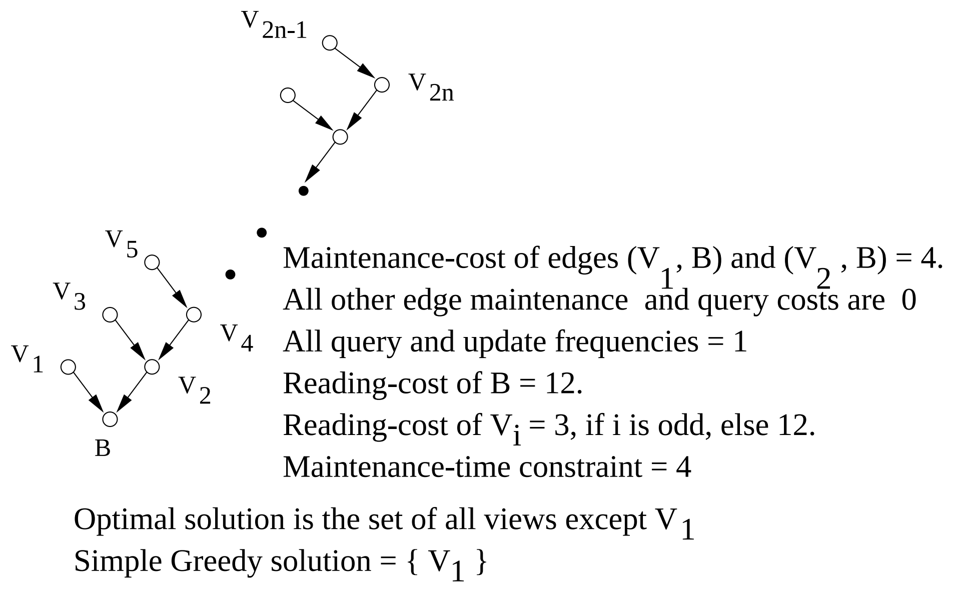
We first compute the query benefit of V_1 at the initial stage when only the base relation B is available (materialized). Recall from Example 1 that $Q(V_i, \phi) = 12$ for all $i \leq 5$ and $Q(B, \phi) = 12$. Thus, $\tau(\mathcal{G}, \phi) = 12 \times 6 = 72$, as all the query frequencies are 1. Also, $Q(V_1, \{V_1\}) = 7$, as the reading-cost of V_1 is 7, $Q(V_i, \{V_1\}) = 12$ for $i = 2, 3, 4, 5$, and $Q(B, \{V_1\}) = 12$. Thus, $\tau(G, \{V_1\}) = 12 \times 5 + 7 = 67$ and thus, the initial query benefit of V_1 is $72 - 67 = 5$. Similarly, the initial query benefits of each of the views V_2, V_3, V_4, and V_5 can be computed to be 4.

Also, $U(\{V_i\}) = UC(V_i, \{V_i\}) = 4$ as the minimum maintenance-length of a path from any V_i to B is 4. Thus, the solution returned by the simple greedy algorithm, that picks the most beneficial view, as defined above, at each stage, is $\{V_1\}$.

It is easy to see that the optimal solution is $\{V_2, V_3, V_4, V_5\}$ with a query benefit of 11 and a total maintenance time of 4. To demonstrate the **nonmonotonic** behavior of the benefit per unit maintenance-cost function, observe that the query-benefits per unit maintenance-cost of sets $\{V_2\}, \{V_3\}, \{V_2, V_3\}$ are 1, 1, and 7/4 respectively. This nonmonotonic behavior is the reason why the simple greedy algorithm that selects views on the basis of their query-benefits per unit maintenance-cost can deliver an arbitrarily bad solution.

Figure 5 shows an extended example where the optimal solution can be made to have an arbitrarily high query benefit, while keeping the simple greedy solution unchanged. Initially, the query benefit of any odd numbered V_i is 12 -3 = 9, while the query benefit of any even numbered V_i is 0. Also, the maintenance cost of any V_i is 4. Hence, the simple greedy algorithm starts by selecting $\{V_1\}$. As the maintenance cost constraint is 4 units, the solution

[12]The effective maintenance-cost may be negative. The results in this section hold nevertheless.



V_{2n-1}
V_{2n}
V_5
V_3
V_4
V_1
V_2
B
Maintenance-cost of edges (V_1, B) and (V_2 , B) = 4.
All other edge maintenance and query costs are 0
All query and update frequencies = 1
Reading-cost of B = 12.
Reading-cost of V_i = 3, if i is odd, else 12.
Maintenance-time constraint = 4
Optimal solution is the set of all views except V_1
Simple Greedy solution = { V_1 }

Figure 5: An OR view graph, $\mathcal{H}$, for which simple greedy performs arbitrarily bad

returned by the simple greedy algorithm is $\{V_1\}$. The optimal solution is $\{V_2, V_3, \ldots, V_{2n}\}$ with a huge query benefit, but a maintenance cost of only 4. □

The above example shows that an appropriate definition of benefit is not enough to guarantee a good solution, and that selecting the most beneficial view at each stage can lead to a very bad greedy strategy.

Note that the nodes in the OR view graphs $\mathcal{G}$ and $\mathcal{H}$, presented in Figure 4 and Figure 5 respectively, can be easily mapped into *real* queries involving aggregations over the base data B. The query-costs associated with the edges in $\mathcal{G}$ and $\mathcal{H}$ depict the *linear cost model*, where the cost of answering a query on v using its descendant u is directly proportional to the size of the view u, which in our model of OR view graphs is represented by the reading-cost of u. Notice that the minimum query-length of a path from u to v in $\mathcal{G}$ or $\mathcal{H}$ is R_v, the reading-cost of v. As zero maintenance-costs in the OR view graphs $\mathcal{G}$ and $\mathcal{H}$ can be replaced by extremely small quantities, the OR view graphs $\mathcal{G}$ and $\mathcal{H}$ depict the plausible scenario when the cost of maintaining a view u from a materialized view v is negligible in comparison to the maintenance cost incurred in maintaining a view u directly from the base data B.

Definition 6 (**Inverted Tree Set**) A set of nodes R is defined to be an inverted tree set in a directed graph G if there is a subgraph (not necessarily induced) T_R in the transitive closure of G such that the set of vertices of T_R is R, and the inverse graph[13] of T_R is a tree.[14]

[13]The inverse of a directed graph is the graph with its edges reversed.

[14]Here, by a tree we mean a *connected* graph wherein each vertex except the root has exactly one incoming edge.

In the OR view graph $\mathcal{G}$ of Figure 4, any subset of $\{V_2, V_3, V_4, V_5\}$ that includes V_2 forms an inverted tree set. Note that $\{V_4, V_5\}$ is forms an inverted tree set. The T_R graph corresponding to the inverted tree set $R = \{V_2, V_3, V_5\}$ has the edges (V_2, V_5) and (V_2, V_3) only. □

The motivation for the inverted tree set comes from the following observation, which we prove in Lemma 5. In an OR view graph, an arbitrary set O (in particular an optimal solution O), can be partitioned into inverted tree sets such that the effective maintenance-cost of O with respect to an already materialized set M is greater than the sum of effective-costs of inverted tree sets with respect to M.

Based on the notion of an inverted tree set, we develop a greedy heuristic called the *Inverted-Tree Greedy Algorithm* which, at each stage, considers all inverted tree sets in the given view graph and selects the inverted tree set that has the most query-benefit per unit effective maintenance-cost.

Algorithm 4 **Inverted-Tree Greedy Algorithm**

```
Given: An OR view graph ($G$), and a total view maintenance time constraint $S$
BEGIN
  $M = \phi$;  $B_C = 0$;
  repeat
    for each inverted tree set of views $T$ in $G$ such that $T \cap M = \phi$
      if ($EU(T, M) \leq S$) and ($B(T, M)/EU(T, M) > \mathrm{B_C}$)
        $B_C = B(T, M)/EU(T, M)$;
        $C = T$;
      end if;
    end for;
    $M = M \cup C$;
  until ($U(M) \geq S$);
  return $M$;
END.                                                                ◇
```

We prove in Theorem 8 that the Inverted-tree greedy algorithm is guaranteed to deliver a near-optimal solution. In Section 6.5, we present experimental results that indicate that in practice, the Inverted-tree greedy algorithm almost always returns an optimal solution. We now define a notion of update graphs which is used to prove Lemma 5.

Definition 7 (**Update Graph**) Given an OR view graph G and a set of nodes/views O in G. An update graph of O in G is denoted by U_O^G and is a subgraph of G such that $V(U_O^G) = O$, and $E(U_O^G) = \{(v, u) \mid u, v \in O$ and $v (\in O)$ is such that $UC(u, \{v\}) \leq UC(u, \{w\})$ for all $w \in O\}$. We drop the superscript G of U_O^G, whenever evident from context. □

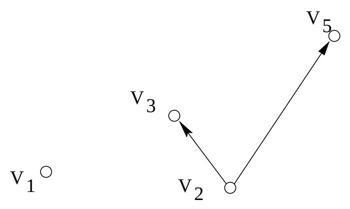


Figure 6: The update graph for $\{V_1, V_2, V_3, V_5\}$ in G

It is easy to see that an update graph is an embedded forest in G. An update graph of O is useful in determining the flow of changes when maintaining the set of views O. An edge (v, u) in an update graph U_O signifies that the view u uses the view v (or tables computed for maintenance of v) to incrementally maintain itself, when the set O is materialized. Figure 6 shows the update graph of $\{V_1, V_2, V_5\}$ in the OR view graph $\mathcal{G}$ of our running example in Figure 4.

Lemma 5 *For a given set of views M, a set of views O in an OR view graph G can be partitioned into inverted tree sets $O_1, O_2, \ldots, O_m$, such that $\sum_{i=1}^{m} EU(O_i, M) \leq EU(O, M)$.* ■

Theorem 8 *Given an OR view graph G and a total maintenance-time constraint S. The Inverted-tree greedy algorithm (Algorithm 4) returns a solution M such that $U(M) \leq 2S$ and M has a query benefit of at least $(1 - 1/e) = 63\%$ of that of an optimal solution that has a maintenance cost of at most $U(M)$, under the assumption that the optimal solution doesn't have an inverted tree set O_i such that $U(O_i) > S$.*

Proof: It is easy to see that the maintenance cost of the solution returned by the Inverted-tree greedy algorithm is at most $2S$ units. Let O be a solution having $U(M) = k$ units of total maintenance time, with an optimal query benefit of B.

Consider a stage when the Inverted-tree greedy algorithm has already chosen a set M having a total maintenance time of l units with incremental per unit query benefits of $a_1, a_2, \ldots, a_l$. Thus, the absolute query benefit of M, $B(M, \phi)$, is $\sum_{i=1}^{l} a_i$. Trivially, the query benefit of the set $O \cup M$ is at least B. Therefore, the query benefit $B(O, M)$ of the set O with respect to M is at least $B - \sum_{i=1}^{l} a_i$.

By Lemma 5, the set O can be partitioned into inverted tree sets $O_1, O_2, \ldots, O_m$ such that $\sum_{i=1}^{m} EU(O_i, M) \leq EU(O, M)$. Also, by monotonicity of the query-benefit function, $B(O, M) \leq \sum_{i=1}^{m} B(O_i, M)$. Now, it is easy to show by contradiction that there is an inverted tree set view O_i such that $B(O_i, M)/EU(O_i, M) \geq B(O, M)/EU(O, M)$, i.e., the query-benefit per unit of effective maintenance-cost of O_i is at least that of O at this stage (else $B(O, M) > \sum_{i=1}^{m} B(O_i, M)$).

As $EU(O_i, M) \leq U(O_i) \leq S$, O_i is also considered for selection by the Inverted-tree greedy at this stage. Thus, the query benefit per unit of effective maintenance-cost of the set C selected by the Inverted-tree algorithm is at least the query-benefit per unit effective maintenance-cost of O_i at this stage. Now as $EU(O,M) \leq k$, we have $B(C,M)/EU(C,M) \geq B(O_i,M)/EU(O_i,M) \geq B(O,M)/EU(O,M) \geq B(O,M)/k \geq (B - \sum_{i=1}^{l} a_i)/k$. 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 $0 < j \leq EU(C,M)$. As the above analysis is true for each set C selected at any stage, we have

$$B \leq ka_j + \sum_{i=1}^{j-1} a_i, \qquad \text{for } \ 0 < j \leq k.$$

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 $A = \sum_{i=1}^{k} a_i = B(M,\phi)$, the (absolute) query benefit of M. ∎

The simplifying assumption made in the above algorithm is almost always true, because $U(M)$ is not expected to be much higher than S. The following theorem (see [14] for proof) proves a similar performance bound without the assumption used in Theorem 8.

Theorem 9 *Given an OR view graph G and a total maintenance-time constraint S. The Inverted-tree greedy algorithm (Algorithm 4) returns a solution M such that $U(M) \leq 2S$ and $B(M,\phi)/U(M) \geq 0.5B(O,\phi)/S$, where O is an optimal solution such that $U(O) \leq S$.* ∎

Dependence of Query and Update Frequencies. Note that we have not made any assumptions about the independence of query frequencies and update frequencies of views. In fact, the query frequency of a view may decrease with the materialization of other views. It can be shown that the above performance guarantees hold even when the query frequency of a view decreases with the materialization of other views.

Time Complexity. In the worst case the total number of inverted tree sets in an OR view graph G is exponential in the size of the graph. However, it is not difficult to see that for the special case of an OR view graph being a *balanced binary tree*, each stage of the Inverted-tree greedy algorithm runs in polynomial time $O(n^2)$, where n is the number of nodes in the graph.

Even though the worst-case time complexity of the Inverted-tree greedy algorithm for general OR view graphs is exponential, our experiments (Section 6.5) show that the Inverted-tree greedy approach takes substantially less time than the A^* algorithm, especially for sparse graphs. Also, the space requirements of the Inverted-tree greedy algorithm is polynomial in the size of graph while that of the A^* heuristic is exponential in the size of the input graph.

6.4 A^* Heuristic

In this section, we present an A^* heuristic that, given an AND-OR view graph and a quantity S, deliver a set of views M that has an optimal query response time such that the total maintenance cost of M is less than S. Recollect that an A^* algorithm [16] searches for an optimal solution in a search graph where each node represents a candidate solution.

Let G be an AND-OR view graph instance and S be the total maintenance-time constraint. We first number the set of views (nodes) N of the graph in an inverse topological order $<v_1, v_2, \ldots, v_n>$ so that all the edges (v_i, v_j) in G are such that $i > j$. We use this order of views to define a binary tree T_G of candidate feasible solutions, which is the search tree used by the A^* algorithm to search for an optimal solution. Each node x in T_G has a label $<N_x, M_x>$, where $N_x = \{v_1, v_2, \ldots, v_d\}$ is a set of views that have been considered for possible materialization at x and $M_x (\subset N_x)$, is the set of views chosen for materialization at x. The root of T_G has the label $<\phi, \phi>$, signifying an empty solution. Each node x with a label $<N_x, M_x>$ has two successor nodes $l(x)$ and $r(x)$ with the labels $<N_x \cup \{v_{d+1}\}, M_x>$ and $<N_x \cup \{v_{d+1}\}, M_x \cup \{v_{d+1}\}>$ respectively. The successor $r(x)$ exists only if $M_x \cup \{v_{d+1}\}$ has a total maintenance cost of less than S, the given cost constraint.

The Algorithm 5 shown below depicts the A^* heuristic for the maintenance-cost view-selection problem in general AND-OR graphs. We define two functions[15] $g : V(T_G) \mapsto \mathcal{R}$, and $h : V(T_G) \mapsto \mathcal{R}$, where $\mathcal{R}$ is the set of real numbers. For a node $x \in V(T_G)$, with a label $<N_x, M_x>$, the value $g(x)$ is the total query cost of the queries on N_x using the selected views in M_x. That is,

$$g_x = \sum_{v_i \in N_x} f_{v_i} Q(v_i, M_x).$$

The number $h(x)$ is an estimated lower bound on $h^*(x)$ which is defined as the remaining query cost of an optimal solution corresponding to some descendant of x in T_G. In other words, $h(x)$ is a lower bound estimation of $h^*(x) = \tau(G, M_y) - g(x)$, where M_y is an optimal solution corresponding to some descendant y of x in T_G.

Algorithm 5 <u>A^* **Heuristic**</u>
Input: G, an AND-OR view graph, and S, the maintenance-cost constraint.
Output: A set of views M selected for materialization.

```
BEGIN
  Create a tree $T_G$ having just the root $A$. The label associated with $A$ is $<\phi, \phi>$.
  Create a priority queue (heap) $L = <A>$.
  repeat
    Remove $x$ from $L$, where $x$ has the lowest $g(x) + h(x)$ value in $L$.
    Let the label of $x$ be $<N_x, M_x>$, where $N_x = \{v_1, v_2, \ldots, v_d\}$ for some $d \leq n$.
    if $(d = n)$ return $M_x$.
    Add a successor of $x$, $l(x)$, with a label $<N_x \cup \{v_{d+1}\}, M_x>$ to the list $L$.
    if $(U(M_x) < S)$
```

[15]The function g should not be confused with the update frequency g_v of a view in a view graph.

```
        Add to $L$ a successor of $x$, $r(x)$, with a label $<N_x \cup \{v_{d+1}\}, M_x \cup \{v_{d+1}\}>$.
    until ( $L$ is empty);
    return NULL;
END.                                                                                    ◇
```

We now show how to compute the value $h(x)$, a lower bound for $h^*(x)$, for a node x in the binary tree T_G. Let $N = V(G)$ be the set of all views/nodes in G. Given a node x, we need to estimate the optimal query cost of the remaining queries in $N - N_x$. Let $s(v) = g_v UC(v, N)$, the minimum maintenance time a view v can have in presence of other materialized views. Also, if a node $v \in V(G)$ is not selected for materialization, queries on v have a minimum query cost of $p(v) = f_v Q(v, N - \{v\})$. Hence, for each view v that is not selected in an optimal solution M_y containing M_x, the remaining query cost accrues by at least $p(v)$. Thus, we fill up the remaining maintenance time available $S - U(M_x)$ with views in $N - N_x$ in the order of their $p(v)/s(v)$ values. The sum of the $f_v Q(v, N - \{v\})$ values for the views left out will give a lower bound on $h^*(x)$, the optimal query cost of the remaining queries. The above described algorithm to compute $h(x)$ is presented formally in Appendix B, along with the proof of the following theorem.

Theorem 10 *The A^* algorithm (Algorithm 5) returns an optimal solution.* ■

The above theorem guarantees the correctness of A^* heuristic. Better lower bounds yield A^* heuristics that will have better performances in terms of the number of nodes explored in T_G. In the worst case, the A^* heuristic can take exponential time in the number of nodes in the view graph. There are no better bounds known for the A^* algorithm in terms of the function $h(x)$ used.

6.5 Experimental Results

We had run some experiments to determine the quality of the solution delivered and the time taken in practice by the Inverted-tree Greedy algorithm for OR view graphs. We ran our algorithms on random acyclic OR view graphs with varying edge-densities. A random directed acyclic is generated by tossing a biased coin to decide whether an edge exists between a pair of nodes. The random bias gives the edge-density of the generated graph. For each randomly generated instance of a view graph, we labeled the nodes with random query and update frequencies. We start by describing the cost model we used for the purposes of our experiments.

Cost Model. For the purposes of experimentation, we assumed that each view is an aggregate view defined over the base data. Hence, we assume a linear query cost model, wherein the cost of answering a query u from its descendant v in a view graph is proportional to $|v|$, the size of v. The experimental results are independent of the proportionality factor(s). The linear cost model is a reasonable assumption when each view in the OR graph is an aggregate view.

For the purposes of computing maintenance costs, we assume the following model for cost of maintaining a view u in presence of a descendant v. The changes to u, Δu, can be computed from changes to v, Δv, in time proportional to $|\Delta v|$, and the view u can be refreshed using Δu in time proportional to $|\Delta u|$. Thus, the total maintenance time incurred in maintaining u using its materialized descendant v is proportional to $(|\Delta u| + |\Delta v|)$. For sake of simplicity, we further assume that $(|\Delta u| + |\Delta v|)$ is proportional to $(|u| + |v|)$ as is likely to be the case when updates are insertion generating (as defined in [17]), or when the updates are update generating but uniformly spread across the domain.

Also, as we are considering aggregate views, we assigned random sizes to each view/node in the view graph in such a way that the size of a view u was less than the size of each of its descendants.

Observations. In our experiments, the Inverted-tree Greedy Algorithm (Algorithm 4) returned an optimal solution as computed by the A^* heuristic for almost all (96%) view graph instances. In other cases, the solution returned by the Inverted-tree greedy algorithm had a query benefit of around 95% of the optimal query benefit.

For balanced trees and sparse graphs having edge density less than 40%, the Inverted-tree greedy took substantially less time (a factor of 10 to 500) than that taken by the A^* heuristic. With the increase in the edge density, the benefit of Inverted-tree greedy over the A^* heuristic reduces and for very dense graphs, A^* may actually perform marginally better than the Inverted-tree greedy. One should observe that OR view graphs that are expected to arise in practice would be very sparse. For example, the the OR view graph corresponding to a data cube having n dimensions has $\sum_{i=1}^{n}(\binom{n}{i}2^i) = 3^n$ edges and 2^n vertices. Thus, the edge density is approximately $(0.75)^n$, for a given n.

As the space requirement of an A^* heuristic grows exponentially in the size of the input graph, we could not run A^* heuristic for N larger than around 25 because of memory-space limitations. In contrast, note that the Inverted-Tree Greedy heuristic takes only quadratic $O(n^2)$ space, where n is the number of views in an OR view graph. In light of the performance guarantees (Theorem 8 and Theorem 10) of the algorithms, our experiment results provide a good evidence for efficiency of our proposed inverted-tree greedy algorithm.

Explanation of Figures. The comparison of the time taken by the Inverted-tree greedy and the A^* heuristic is presented in Figure 7. In all the plots shown in Figure 7, the different view graph instance s of the maintenance-cost view-selection problem are plotted on the x-axis. A view graph instance G is represented in terms of N, the number of nodes in G, and S, the maintenance-time constraint. The view graph instances are arranged in the lexicographic

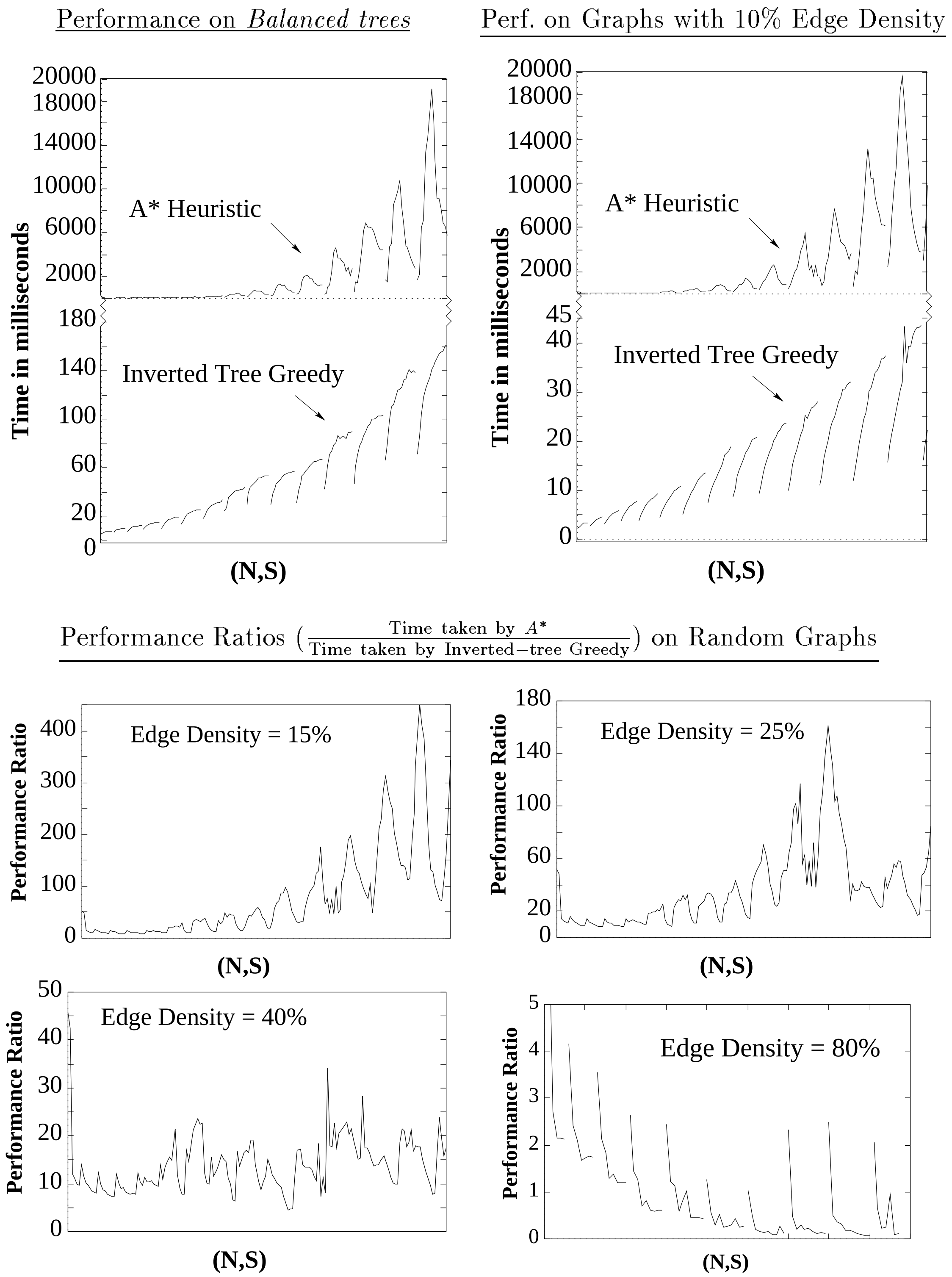


Figure 7: Experimental results. The x-axis shows the view graph instances in lexicographic order of their (N, S) values, where N is the number of nodes in the graph and S is the maintenance-time constraint.

order of (N, S), i.e., all the view graphs with smallest N are listed first, in order of their constraint S values. In all the graph plots, the number N varies from 10 to 25, and S varies from the time required to maintain the smallest view to the time required to maintain all views in a given view graph. The breaks in the graph plots depict a change in the value of N.

In Figure 7, for the case of balanced trees and view graphs with an edge-density of 10%, we have plotted times taken by the Inverted-tree greedy as well as the A^* heuristic. One can see that the time taken by A^* heuristic is 100 to 500 times the time taken by the Inverted-tree greedy. For other graphs instances of edge densities 15%, 25%, and 40%, we have plotted the performance ratio (the ratio of the time taken by the A^* heuristic to the time taken by the Inverted-tree Greedy.) We also ran experiments on random graphs with number of edges linear in the number of nodes. The performance ratio obtained for random graphs having $2n$ edges, where n is the number of nodes in the graph, was similar to the case of 15% edge density.

For a particular value of N, the time taken by the A^* heuristic first increases and then decreases, with increase in S, the maintenance-time constraint. The initial increase is due to the increase in the number of feasible solutions and the later decrease is due to the fact that $h(x)$ becomes a better approximation of $h^*(x)$ with the increase S.

7 Related Work

In the initial research done on the view-selection problem, Harinarayan, Rajaraman, and Ullman [10] presented algorithms for the view-selection problem in data cubes under a disk-space constraint. A data cube is a special purpose data warehouse, where there are only queries with aggregates over the base relation. Gupta et al. extended their work to include indexes in [11]. The work presented in this article is an extension and generalization of the work in [10], and first appeared in [15] and [18].

There has also been some negative theoretical results on the problem of selection of views. In particular, Karloff et al. [19] show that the variation of the view-selection problem where the goal is to optimize the query cost (in contrast to our optimization goal of query benefit) is inapproximable for general partial orders. In other work, Chirkova et al. in [20] show that the number of views involved in an optimal solution for the view-selection problem may be exponential in the size of the database schema, when the query optimizer has good estimates of the sizes of the views. The result was lated extended in [21] for a more practical query cost model. In both articles ([20, 21]), the authors assume that the input consists of only the database scheme and the workload queries, whereas our approximation algorithms start with a query-view graph as an input.

Apart from the above theoretical research, there has been a substantial amount of effort

([22, 23, 24, 25]) on developing heuristics for the view-selection problem that may work well in practice. Most of the work done in this context has developed various frameworks and heuristics for selection of views in order to optimize the query response time and/or view maintenance time with or without a resource constraint. The heuristics developed are either exhaustive searches or do not have any performance guarantees on the quality of the solution delivered. In the similar vein, [26, 27, 28] have used randomized and genetic approaches to select a set of views to materialize. In particular, Kalnis et al. in [28] show that randomized search methods provide near-optimal solutions and can easily be adapted to various versions of the problem, including existence of size and time constraints. In validation of our greedy approach, they also show through experiments that the greedy algorithms delivers near-optimal solutions. Our work presented in this article differs for the above works in that we have designed approximation algorithms that deliver a *provably* good solution with a query benefit within a constant factor of the optimal query benefit.

There has also been a fair amount of work in incorporating various heuristics for view-selection problem into commercial database systems. Most of the work done in this context use some variation of the greedy heuristic for selection of views and/or indexes. In particular, Chaudhury and Narasayya [29] use a variation of the greedy approach, wherein they select an optimal 'seed' of k indexes as a starting point, and show that the greedy heuristic with a seed of size 2 does very well for a large variety of workloads over Microsoft SQL Server. Some other systems, e.g., Redbrick/Informix Vista [30] and Oracle 8i, also provide tools to tune the selection of materialized views for a workload, wherein they use variations of the greedy heuristics for selection of views. Also, Agrawal et al. in [31], extending their earlier work ([29, 32]), describe an architecture for selection of views and indexes, where they use various ways of effectively pruning the space of possible views and indexes.

8 Concluding Remarks

A data warehouse is built for the purposes of information integration and/or decision support and analysis. One of the most important design issues that arise in a data warehouse is selection of views to materialize. This article has extensively addressed the problem of selection of views and made significant contributions in solving it comprehensively.

In particular, we have developed a theoretical framework for the general view-selection problem of selection of views in a data warehouse. We have presented polynomial-time greedy heuristics that provably deliver a solution within a constant factor of the optimal for some important special cases, viz. OR view graphs and AND view graphs, under the disk-space constraint. For each of these special cases, we have extended the algorithms to graphs when there are indexes associated with each view and proved approximation bounds for the extended settings. The developed heuristic was also extended to the most general case of AND-OR view graphs. Finally, we addressed the maintenance-cost view-selection problem where the resource constraint is the total maintenance cost of the views selection for materialization. For the maintenance-cost view-selection problem, we developed a inverted-tree greedy heuristic that provably delivers a competitive solution for the OR view graphs. Since, the designed inverted-tree greedy heuristic is exponential in worst case analysis, we show empirically that the inverted-tree heuristic performs much better than the alternative A^* heuristic and almost always returns an optimal solution. Our results are summarized in Table 1.

Algorithm	**View Graphs**	**Constraint**	**Performance**	**Complexity**
Greedy	OR/AND	Disk Space	63%	$O(kn^2)$
Inner-level Greedy	OR/AND with indexes	Disk Space	47%	$O(k^2n^2)$
AO-Greedy	AND-OR	Disk Space	63%	Exponential†
r-level Greedy	AND-OR	Disk Space	$g(r)$‡	$O((kn)^{2r})$
Inverted-tree Greedy	OR	Update Cost	63%	Exponential§
A^*	AND-OR	Update Cost	Optimal	Exponential

Table 1: Algorithms. Here, n is the size of the view-graph, and k is the constraint.

The techniques developed in this article offer significant insights into the greedy heuristic and in particular, into the nature of the view-selection problem in a data warehouse. The techniques developed in this article seem to have much broader applications. Recently, we have applied some of the techniques developed in this article to the problem of computing a dominating set in hypergraphs [33] in the context of exploiting data correlation in sensor networks, and to the problem of selecting an optimal set of intermediate nodes to store documents in ad hoc wireless networks [34].

There are still a lot of important open questions in the context of the view-selection problem. In particular, there is still very little known on the approximability of the view-selection problem in general AND-OR view graphs. Also of significant interest is to design approximation algorithms for other special cases of AND-OR view graphs viz. binary AND-OR view trees, AND-OR graphs that arise in aggregate views with selection and union, and AND-OR view graphs for range queries/views which can be mapped to rectangles in a two dimensional plane. Moreover, the view-selection problem in AND view graphs is not yet known to be NP-hard.

†AO-Greedy algorithm is exponential in the number of edges in the intersection graph F_ζ.

‡The value $g(r)$ is defined recursively as $g(r) = 1 - 1/e^{g(r-1)}$, and $g(0) = 1$.

§Inverted-tree greedy algorithm is polynomial in the number of inverted tree sets.

References

[1] S. Chaudhuri and K. Shim, "Including groupby in query optimization," in *Proceedings of the International Conference on Very Large Database Systems*, 1994.

[2] A. Gupta, V. Harinarayan, and D. Quass, "Generalized projections: A powerful approach to aggregation," in *Proceedings of the Intl. Conference on Very Large Database Systems*, 1995.

[3] W. Yan and P. Larson, "Eager aggregation and lazy aggregation," in *Proceedings of the Twenty-First International Conference on Very Large Databases (VLDB)*, pp. 345–357, 1995.

[4] P. O'Neil and G. Graefe, "Multi-table joins through bitmapped join indices," *SIGMOD Record*, vol. 24, no. 3, pp. 8–11, 1995.

[5] J. Widom, "Research problems in data warehousing," in *Proceedings of the Fourth International Conference on Information and Knowledge Management*, 1995.

[6] R. Kimball, *The Data Warehouse Toolkit.* John Wiley & Sons, Inc., 1996.

[7] N. Roussopoulos, "The logical access path schema of a database," *IEEE Transaction in Software Engineering*, vol. SE-8, pp. 563–573, Nov. 1982.

[8] T. K. Sellis, "Multiple query optimization," *ACM Trans. on Database Systems*, vol. 13, 1988.

[9] A. Cosar, E.-P. Lim, and J. Srivastava, "Multiple query optimization with depth-first branch-and-bound and dynamic query ordering," in *CIKM*, 1993.

[10] V. Harinarayan, A. Rajaraman, and J. Ullman, "Implementing data cubes efficiently," in *Proceedings of the ACM SIGMOD International Conference of Mangement of Data*, 1996.

[11] H. Gupta, V. Harinarayan, A. Rajaraman, and J. Ullman, "Index selection in OLAP," in *Proceedings of the International Conference on Data Engineering*, 1997.

[12] U. Feige, "A threshold of ln n for approximating set cover," in *Proc. of the ACM STOC*, 1996.

[13] U. S. Chakravarthy and J. Minker, "Processing multiple queries in database systems," *Database Engineering*, vol. 5, pp. 38–44, Sept. 1982.

[14] H. Gupta, *Selection and Maintenance of Materialized Views in a Data Warehouses.* PhD thesis, Stanford University, Department of Computer Science, 1999.

[15] H. Gupta, "Selection of views to materialize in a data warehouse," in *Proc. of ICDT*, 1997.

[16] N. Nilsson, *Principles of Artificial Intelligence.* Morgan Kaufmann Publishers, Inc., 1980.

[17] I. Mumick, D. Quass, and B. Mumick, "Maintenance of data cubes and summary tables in a warehouse," in *Proceedings of the ACM SIGMOD Intl. Conference of Mangement of Data*, 1997.

[18] H. Gupta and I. Mumick, "Selection of views to materialize under a maintenance cost constraint," in *Proceedings of the International Conference on Database Theory*, 1999.

[19] H. Karloff and M. Mihail, "On the complexity of the view-selection problem," in *PODS*, 1999.

[20] R. Chirkova, A. Halevy, and D.Suciu, "A formal perspective on the view selection problem," in *Proceedings of the International Conference on Very Large Database Systems*, 2001.

[21] R. Chirkova, "The view selection problem has an exponential bound for conjunctive que ries and views," in *Proceedings of the ACM Symposium on Principles of Database Systems*, 2002.

[22] J. Yang, K. Karlapalem, and Q. Li, "Algorithms for materialized view design in data warehousing environment," in *Proceedings of the Intl. Conference on Very Large Database Systems*, 1997.

[23] E. Baralis, S. Paraboschi, and E. Teniente, "Materialized view selection in a multidimensional database," in *Proceedings of the Intl. Conference on Very Large Database Systems*, 1997.

[24] D. Theodoratos and T. Sellis, "Data warehouse configuration," in *Proc. of VLDB*, 1997.

[25] A. Shukla, P. Deshpande, and J. Naughton, "Materialized view selection for multidimensional datasets," in *Proceedings of the International Conference on Very Large Database Systems*, 1998.

[26] C. Zhang and J. Yang, "Genetic algorithm for materialized view selection in data warehouse environments," in *Intl. Conf. on Data Warehousing and Knowledge Discovery (DaWaK)*, 1999.

[27] M. Lee and J. Hammer, "Speeding up materialized view selection in data warehouses using a randomized algorithm," *Intl. Journal of Cooperative Information Systems*, vol. 10, no. 3, 2001.

[28] P. Kalnis, N. Mamoulis, and D. Papadias, "View selection using randomized search," *Data and Knowledge Engineering*, vol. 42, no. 1, 2002.

[29] S. Chaudhuri and V. Narasayya, "An efficient cost-driven index selection tool for microsoft sql server," in *Proceedings of the International Conference on Very Large Database Systems*, 1997.

[30] L. Colby et al., "Redbrick vista: Aggregate computation and management," in *ICDE*, 1998.

[31] S. Agrawal, S. Chaudhuri, and V. Narasayya, "Automated selection of materialized views and indexes in sql databases," in *Proc. of the Intl. Conf. on Very Large Database Systems*, 2000.

[32] S. Chaudhuri and V. Narasayya, "Microsoft index tuning wizard for sql server 7.0," in *Proceedings of the ACM SIGMOD Intl. Conf. of Mangement of Data*, 1998.

[33] H. Gupta, V. Navda, S. Das, and V. Chowdhary, "Energy-efficient gathering of correlated data in sensor networks," tech. rep., SUNY, Stony Brook, 2003.

[34] B. Tan, S. Das, and H. Gupta, "Selection of intermediate storage nodes in ad hoc networks," tech. rep., SUNY, Stony Brook, 2003.

A Proofs of Lemmas

Lemma 1 *In an AND view graph, $B(v, \phi) \geq B(v, M)$ for any view v and a set of views M, if the update frequency g_x at any view x is less than its query frequency f_x.*

Proof: Let A be the set of ancestors of v, including v, in the AND view graph G. Let $M_A = M \cap A$. Let A_D be the set of those ancestors of v which do not have any descendants in the set M_A. For any $x \in A$, we have $Q(x, \phi) - Q(x, v) = Q(v, \phi)$. Therefore,

$$\begin{aligned} B(v,\phi) &= \sum_{x \in A} f_x(Q(x,\phi) - Q(x,v)) - g_v UC(v,v) \\ &= \sum_{x \in A} f_x Q(v,\phi) - g_v UC(v,v) && \text{as } x \in A. \end{aligned}$$

Now consider $B(v, M)$. When M has already been materialized, v reduces then query costs of only the nodes in A_D. Also, materialization of v also helps in reducing the maintenance costs of nodes in M_A. Therefore,

$$\begin{aligned} B(v,M) &= \sum_{x \in A_D} f_x(Q(x,M) - Q(x, M \cup \{v\})) - g_v UC(v, M \cup \{v\}) \\ &\quad + \sum_{x \in M_A} g_x(UC(x,M) - UC(x, M \cup \{v\})) \\ \text{Now} \quad Q(v,\phi) &\geq Q(v,M) \geq UC(x,M) - UC(x, M \cup \{v\}) && \text{for any } x \in M_A, \\ \text{and} \quad Q(v,M) &= Q(x,M) - Q(x, M \cup \{v\}) && \text{for any } x \in A_D. \\ \text{Thus,} \quad B(v,M) &\leq \sum_{x \in A_D} f_x Q(v,M) - g_v UC(v,M) + \sum_{x \in M_A} g_x Q(v,\phi) \end{aligned}$$

$$\leq \sum_{x \in A_D} f_x Q(v,M) - g_v UC(v,M) + \sum_{x \in M_A} f_x Q(v,\phi) \qquad \text{as } g_x \leq f_x.$$

$$\begin{aligned}
B(v,\phi) - B(v,M) \geq\ & \sum_{x \in A} f_x Q(v,\phi) - g_v UC(v,v) \\
& \quad - \sum_{x \in A_D} f_x Q(v,M) + g_v UC(v,M) - \sum_{x \in M_A} f_x Q(v,\phi) \\
\geq\ & \sum_{x \in A} f_x Q(v,\phi) - \sum_{x \in A_D} f_x Q(v,M) - \sum_{x \in M_A} f_x Q(v,\phi) \\
& \quad - g_v(UC(v,v) - UC(v,M)) \\
\geq\ & \sum_{x \in A_D} f_x Q(v,\phi) + \sum_{x \in M_A} f_x Q(v,\phi) - \sum_{x \in A_D} f_x Q(v,M) \\
& \quad - \sum_{x \in M_A} f_x Q(v,\phi) - g_v(UC(v,v) - UC(v,M)), \text{ as } A_D \cap M_A = \phi. \\
\geq\ & \sum_{x \in A_D} f_x (Q(v,\phi) - Q(v,M)) - g_v(UC(v,v) - UC(v,M)) \\
\geq\ & f_v(Q(v,\phi) - Q(v,M)) - f_v(UC(v,v) - UC(v,M))
\end{aligned}$$

Now, let C_{MD} be the cost of materialization all descendants of v that are in M. Then,

$$Q(v,\phi) - Q(v,M) = C_{MD} \geq UC(v,v) - UC(v,M).$$

Therefore, we get $B(v,\phi) - B(v,M) \geq 0$. ■

Lemma 2 *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: Consider views $V_1, V_2, \ldots, V_m$ and a set of views M. Also, for simplicity, let $M_i = M \cup \{V_1, V_2, \ldots, V_i\}$ for $1 \leq i \leq m$. Note that Lemma 1 implies that $B(v,L) \geq B(v, L \cup M)$ for any view v and sets of views L and M. Therefore, we have

$$B(V_i, M) \geq B(V_i, M_i) \text{ for } 1 \leq i \leq m.$$

Also, by definition of the benefit function, we have

$$B(\{V_1, V_2, \ldots, V_m\}, M) = B(\{V_1\}, M) + B(\{V_2\}, M_1) + B(\{V_3\}, M_3) + \ldots + B(\{V_m\}, M_m).$$

Using the above two equations, we get

$$B(\{V_1, V_2, \ldots, V_m\}, M) \leq B(\{V_1\}, M) + B(\{V_2\}, M) + B(\{V_3\}, M) + \ldots + B(\{V_m\}, M),$$

which proves the monotonicity of the benefit function for an arbitrary M with respect to arbitrary views $V_1, V_2, \ldots, V_m$. ■

Lemma 3 *An optimal solution O of the view-selection problem in query-view graph $G = (\zeta \cup Q, E)$ can be partitioned into sets of views $O_1, O_2, \ldots, O_m$, such that each O_i corresponds to a connected subgraph in F_ζ, as defined above, and $B(O,M) \leq \sum_{i=1}^{m} B(O_i, M)$.*

Proof: We start by showing that there exists a subset Γ of ζ such that $O = \bigcup_{\sigma \in \Gamma} \sigma$, if O is an optimal set.

Let Γ be a maximal subset of ζ such that for every $\sigma \in \Gamma$, $\sigma \subseteq O$. Consider an arbitrary view $v \in O$. As O is optimal, v helps answer some query, else it could be removed from O. Thus, for some $\sigma_v \subseteq O$, $v \in \sigma_v \in \zeta$, implying that $\sigma_v \in \Gamma$. Thus, $v \in O$ implies $v \in \bigcup_{\sigma \in \Gamma} \sigma$ for an arbitrary v. Therefore, $O \subseteq \bigcup_{\sigma \in \Gamma} \sigma$. Also, by definition of Γ, it is obvious that $\bigcup_{\sigma \in \Gamma} \sigma \subseteq O$. Hence, $O = \bigcup_{\sigma \in \Gamma} \sigma$.

Now, consider the intersection graph F_Γ of Γ. The intersection graph F_Γ is only an induced subgraph of the intersection graph F_ζ on the nodes in Γ. Consider the connected components of F_Γ which partition the set of nodes Γ into $\Gamma_1, \Gamma_2, \ldots, \Gamma_m$. Let $O_i = \bigcup_{\sigma \in \Gamma_i} \sigma$. Now O_i's also form a partition of O, because there are no edges in F_ζ between the nodes of Γ_i and Γ_j for any i and j. It is easy to see that for the above O_i's, $B(O, M) \leq \sum_{i=1}^{m} B(O_i, M)$, because exactly one materialized node in ζ is used to answer any query q in Q. ∎

Lemma 4 *The InnerGreedy function with the first parameter value equal to r delivers a solution U whose benefit per unit space is at least $g(r)$ of the optimal benefit per unit space achievable. The function $g(r)$ is defined recursively as $g(r) = 1 - 1/e^{g(r-1)}$, and $g(0) = 1$.*

Proof: We prove this lemma by induction. The base case for $r = 0$ is obvious. Assume that the value of the first parameter to the InnerGreedy function is r.

Without loss of generality, we assume that the input parameter M to InnerGreedy is ϕ. Consider a set of views O' in Γ that has the optimal benefit per unit space. It is obvious that O' contains the view v that is in all elements of Γ. Let $O' - \{v\} = O$.

Consider a stage at which the InnerGreedy algorithm has already chosen a set G_l (apart from v) occupying l units of space with incremental benefits $a_1, a_2, a_3 \ldots . . a_l$ with respect to v. Let $G_l^v = G_l \cup \{v\}$, also the value of U (see Algorithm 3) at this stage. The benefit of the set $O \cup G_l$ with respect to $\{v\}$ is at least that of O with respect to v, i.e., $B(O \cup G_l, \{v\}) \geq B(O, \{v\})$. Also, $B(O \cup G_l, \{v\}) = B(O, G_l^v) + \sum_{i=1}^{l} a_i$. Therefore, the benefit of the set O with respect to G_l^v, $B(O, G_l^v)$, is at least $B(O, \{v\}) - \sum_{i=1}^{l} a_i$.

As Γ consists of m connected components $\Gamma_1, \ldots, \Gamma_m$ after deleting v, the set O can be split into m disjoint sets $O_1, O_2, \ldots, O_m$, such that each O_i belongs to Γ_i. By the monotonicity property of the benefit function w.r.t. the sets $O_1, \ldots, O_m$, $B(O, G_l^v) \leq \sum_{i=1}^{m} B(O_i, G_l^v)$. Now, it is easy to show by contradiction that there exists at least one O_i such that $B(O_i, G_l^v)/S(O_i) \geq B(O, G_l^v)/S(O)$ (else $B(O, G_l^v) > \sum_{i=1}^{m} B(O_i, G_l^v)$).

Now, by inductive hypothesis, the benefit per unit space of the set J, selected by the InnerGreedy algorithm at this stage, is at least $g(r-1)$ times $B(O_i, G_l^v)/S(O_i)$, as the InnerGreedy function when called with the first parameter equal to $r - 1$ returns a solution that is within $g(r-1)$ of the optimal. Thus, $B(J, G_l^v) \geq g(r-1) B(O_i, G_l^v)/S(O_i) \geq g(r-1) B(O, G_l^v)/S(O) \geq g(r-1)(B(O, \{v\}) - \sum_{i=1}^{l} a_i)/S(O)$.

Let us assume, that the InnerGreedy Algorithm continues to select views (apart from v) till it has exhausted S(O) space, and the final set of views selected is G. Using techniques similar to the proof of Theorem 2, it is easy to show that $B(G, \{v\}) = (1 - 1/e^{g(r-1)}) B(O, \{v\}) = g(r) B(O, \{v\})$, as $k' = g(r-1)$ here. Thus, we have

$$\begin{aligned} B(G \cup \{v\}, \phi) &= B(v, \phi) + B(G, \{v\}) \\ &\geq B(v, \phi) + g(r) B(O, \{v\}) \\ &\geq g(r) B(O', \phi) \end{aligned}$$

Thus, the benefit per unit space of $G \cup \{v\}$ is at least $g(r) B(O', \phi)/S(O')$, as G and O occupy the same space. But, the InnerGreedy algorithm actually stops when the benefit of U per unit space reaches the maximum. Therefore, the benefit per unit space of U is at least equal to the benefit per unit space of G, which is $g(r)$ times the optimal. ∎

Lemma 5 *For a given set of views M, a set of views O in an OR view graph G can be partitioned into inverted tree sets $O_1, O_2, \ldots, O_m$, such that $\sum_{i=1}^{m} EU(O_i, M) \leq EU(O, M)$.*

Proof: Consider the update graph U_O of O in G. By definition, U_O is a forest consisting of m trees, say, $U_1, \ldots, U_m$ for some $m \leq |O|$. Let, $O_i = V(U_i)$, for $i \leq m$.

An edge (y, x) in the update graph U_O implies the presence of an edge (x, y) in the transitive closure of G. Thus, an embedded tree U_i in the update graph U_O is an embedded tree in the transitive closure of the inverse graph of G. Hence, the set of vertices O_i is an inverted tree set in G.

For a set of views C, we use $UC(C, M)$ to denote the maintenance cost of the set C w.r.t. M, i.e., $UC(C, M) = \sum_{v \in C} g_v UC(v, M \cup C)$, where $UC(v, M)$ for a view v is the maintenance cost of v in presence of M as defined in Section 2.3. Also, let $Rd(M, C) = U(M) - UC(M, C)$, i.e., the reduction in the maintenance time of M due to the set of views C. Now, the effective maintenance-cost of a set O_i with respect to a set M, $EU(O_i, M)$, can be written as

$$\begin{aligned} EU(O_i, M) &= (UC(O_i, M) + UC(M, O_i)) - U(M) \\ &= UC(O_i, M) - (U(M) - UC(M, O_i)) \\ &= UC(O_i, M) - Rd(M, O_i) \end{aligned}$$

As no view in a set O_i uses a view in a different set O_j for its maintenance,

$$UC(O, M) = \sum_{i=1}^{m} UC(O_i, M).$$

Also, as any view uses at most one view to help maintain itself, the reduction in the maintenance cost of M due to the set O is less than the sum of the reductions due to the sets $O_1, \ldots, O_m$, i.e.,

$$Rd(M, O) \leq \sum_{i=1}^{m} Rd(M, O_i).$$

Therefore, we have $EU(O, M) = UC(O, M) - Rd(M, O) \geq \sum_{i=1}^{m} UC(O_i, M) - \sum_{i=1}^{m} Rd(M, O_i) \geq \sum_{i=1}^{m}(EU(O_i, M)$. ∎

B A^* Heuristic

Algorithm 6 Computing h

```
Input: An AND-OR view graph ($G$), a maintenance-time constraint $S$, and
  a node $x$ with a label $<N_x, M_x>$ in the search tree $T_G$.
  $N = \{v_1, v_2, \ldots, v_n\}$ is the set of all views/nodes in $G$.
Output: The value $h(x)$.
BEGIN
  Let $N_x = \{v_1, v_2, \ldots, v_d\}$ and $N'_x = N - N_x = \{v_{d+1}, \ldots, v_n\}$.
  For each view $v \in N'_x$, define a profit $p(v) = f_v Q(v, N - \{v\})$ and space $s(v)$
    is $g_v U(v, N)$, the minimum possible maintenance cost of $v$.
  $S_x = 0$;
  Let $w$ be the view that has the highest profit in $N'_x$.
  $P_x = p(w)$;  $N'_x = N'_x - \{w\}$;                /* To nullify the 'knapsack' effect. */
  repeat
    Let $v$ be the view with the highest value of $p(v)/s(v)$ in $N'_x$.
    $S_x = S_x + s(v)$;
    $P_x = P_x + p(v)$;
    $N'_x = N'_x - \{v\}$;
  until ( $S_x \geq S - U(M_x)$);
  $h(x) = 0$;
  for $v \in N'_x$
    $h(x) = h(x) + p(v)$;
  return $h(x)$;
END.                                                                    ◇
```

Theorem 10 *The A^* algorithm (Algorithm 5) returns an optimal solution.*

Proof: If an A^* heuristic expands nodes in the increasing order of their $g(x)+h(x)$ values, it is known ([16]) that the first leaf node found by the algorithm corresponds to an optimal solution. Thus, we only need to show that $h(x)$ is indeed a lower bound of $h^*(x)$, i.e., $h(x) \leq h^*(x)$ for all $x \in V(T_G)$.

Consider an optimal feasible solution M_y corresponding to a node y that is a descendant of x in T_G. Each view $v \notin M_y$ adds at least $p(v) = f_v Q(v, N - \{v\})$ units to the remaining query cost. So, $h^*(x)$, the remaining query cost of the optimal solution M_y, is at least $\sum_{v\in(N-(M_y \cup N_x))} p(v) = \sum_{v\in(N-N_x)} p(v) - \sum_{v\in(M_y-N_x)} p(v) = P_x + h(x) - \sum_{v\in(M_y-N_x)} p(v)$, where $h(x) = \sum_{v\in(N-N_x)} p(v) - P_x$, as computed by Algorithm 6. We will show that $P_x \geq \sum_{v\in(M_y-N_x)} p(v)$, which will imply that $h^*(x)$ is at least $h(x)$.

To prove the above claim, note that $U(M_y) \leq S + UC(v, M_y)$ for some $v \in (M_y - M_x)$. As, $U(M_y) \geq U(M_x) + \sum_{u\in(M_y-M_x)}(s(u))$, where $s(u)$ is the minimum possible maintenance cost of u, we get $\sum_{u\in(M_y-M_x)}(s(u)) \leq S + UC(v, M_y) - U(M_x)$. As $v \in M_y - M_x$ and $UC(v, M_y) \geq s(v)$, we have $\sum_{u\in((M_y-M_x)-\{v\})}(s(u)) \leq S - U(M_x)$. Note that P_x, as computed by the Algorithm 6, is such that $P_x - p(w)$ is more than the maximum profit that can be fit in the knapsack of size $S - U(M_x)$. Thus, $P_x - p(w) \geq \sum_{u\in((M_y-N_x)-\{v\})} p(u)$, which implies that $P_x \geq \sum_{u\in(M_y-N_x)} p(u)$. ∎