

A General Framework for the View Selection Problem for Data Warehouse Design and Evolution*

Dimitri Theodoratos
Department of EE&CS
National Technical University of Athens
Greece
dth@dblaboratory.ece.ntua.gr

Mokrane Bouzeghoub
Laboratoire PRISM
Université de Versailles
France
Mokrane.Bouzeghoub@prism.uvsq.fr

ABSTRACT

A Data Warehouse (DW) can be seen as a set of materialized views defined over remote source relations. During the initial design and evolution of a DW, the DW designer is faced, on many occasions, with the problem of selecting views to materialize in the DW. This problem has been addressed for different classes of queries/views, and with different design goals. In this work we unify these approaches in a general framework for materialized view selection for Data Warehousing. We first identify and analyze different design goals. A design goal can be the minimization of a cost function or a constraint of different types. We then define the general view selection problem that aims at satisfying all these goals together. This definition of the problem allows us to deal not only with the static design of a DW, but also with its evolution. We use expression AND/OR dags to represent alternative ways of evaluating multiple queries and views, and subexpression sharing. Our formalism is general enough to allow the representation of complex queries including grouping/aggregation queries, necessary in DW applications. We show how the design goals can be mapped into conditions on expression AND/OR dag structures. Using this mapping, we determine the search space for the general view selection problem, and we discuss algorithms for exploring it. Our approach can be used as is but it can be also applied to particular DW design cases where not all the design goals are required.

1. INTRODUCTION

A Data Warehouse (DW) is a repository of information collected from multiple, possibly heterogeneous, autonomous, distributed databases and other information

*Work supported by the Platon program on bilateral cooperation between French and Greek governments

sources for the purpose of complex querying, analysis and decision support. Data at the DW are selectively collected from the sources, processed in order to resolve inconsistencies, and integrated in advance (at design time) before data loading. DW data are usually organized multidimensionally to support On-line Analytical Processing (OLAP). A DW can be seen as a set of materialized views defined over the source relations. User queries use these materialized views either partially or completely in order to get evaluated. This is performed through partial or complete rewritings [19, 6] of the queries over the materialized views. Evaluating the queries incurs a query evaluation cost. When the source relations change, the materialized views need to be eventually updated. Maintaining the materialized views incurs a view maintenance cost.

During the initial design and evolution of a DW, the DW designer/administrator is, on many occasions, faced with the problem of selecting views to materialize in the DW. This problem has been addressed in the literature for different classes of queries/views and with different design goals. For instance, [13] decides what grouping/aggregation views to precompute in order to minimize the query evaluation cost under a space constraint; [25] selects auxiliary views for minimizing the total view maintenance cost; [39] chooses views for minimizing a combination of the query evaluation and the view maintenance cost; [35] selects PSJ views for minimizing the combined cost under the constraint of the exclusive answerability of the queries from the materialized views; [12] endeavors to minimize the query evaluation cost under a maintenance cost constraint; an incremental view selection approach is presented in [37]; [24, 1, 21, 22, 32] address the problem of making a view set self-maintainable by adding auxiliary views. All these problems are sub-cases of a general problem that we call *the general view selection problem*. Roughly speaking, the general view selection problem is the problem of selecting views for materialization in order to satisfy a number of design goals. Design goals can be the minimization of a cost function (i.e. view maintenance cost) or a constraint (i.e. space restrictions).

Contribution. In this paper we present a framework for the general view selection problem for DW design

and evolution. This framework encompasses as a special case different view selection problems addressed previously. The main contributions of the paper are the following.

- We identify and analyze design goals adopted in different view selection problems for Data Warehousing. A design goal can be the minimization of a cost function or a constraint. Constraints are further classified as system oriented or user oriented.
- We define the general view selection problem that aims at satisfying all these design goals together. The definition of the problem is wide in that it allows us to deal not only with static DW design issues but also with the evolution of the DW (dynamic design issues).
- We use multiexpression AND/OR dags to represent multiple queries, materialized views, source relations, subexpression sharing, alternative ways of evaluating queries using materialized views, and alternative ways of propagating source relation changes. This formalism allows the representation of complex queries including grouping/aggregation queries that are extensively used in Data Warehousing applications.
- We show how the different design goals can be expressed in terms of multiexpression AND/OR dags.
- Using this mapping, we determine the search space for the general view selection problem, and we discuss algorithms for exploring it.
- This work provides a better understanding of the different view selection problems by highlighting their common points. Our approach can be used as is but it can be also applied to particular DW design cases where not all the design goals are necessary.

Related work. There are a number of papers dealing with the view selection problems. These papers, along with their design goals, are cited in Section 2 and in the introduction. The work presented in this paper unifies these approaches in a general framework for view selection. Papers dealing with query evaluation, view maintenance, and multiexpression dags are cited in the respective sections.

Outline. The rest of the paper is organized as follows. The next section presents and analyses different design goals, and states the general view selection problem. In Section 3, we define multiexpression AND/OR dags and their derivatives. In Section 4, after presenting a cost model, we show how the design goals can be mapped into conditions over multiexpression AND/OR dags. Section 5 defines the search space and discusses algorithms. We conclude in Section 6.

2. DESIGN GOALS AND THE GENERAL VIEW SELECTION PROBLEM

Works addressing view selection problems attempt to satisfy one or more of the design goals presented in this section. A goal is either the minimization of a cost function or a constraint. Further a constraint can be user oriented or system oriented. Clearly, attempting to satisfy the constraints can result in no feasible solution to a view selection problem. Design goals can also be

viewed as quality factors of the DW [16, 5].

2.1 Minimization of cost functions

Most approaches comprise in their design goals the minimization of a cost function.

Minimization of the query evaluation cost. Many view selection problems take as input the queries that the DW has to satisfy for an initial or an incremental design [13, 11, 39, 2, 35, 12, 37, 7]. The overall query evaluation cost is the sum of the cost of evaluating each input query rewritten (partially or completely) over the materialized views. This sum can also be weighted, each weight indicating the frequency, or importance, of the associated query. The approaches [13, 30, 12] aim at minimizing the query evaluation cost.

Minimization of the view maintenance cost. The materialized views are usually maintained using an incremental strategy. In an incremental strategy, only the changes that must be applied to the view are computed using the changes of the source relations [4, 8, 15, 23]. These view changes are then applied to the materialized view. The view maintenance cost is the sum of the cost of propagating each source relation change to the materialized views. This sum can be weighted, each weight indicating the frequency of propagation of the changes of the associated source relation. The expressions used to compute the changes to be applied to a view involve the changes of the source relations, and are called maintenance expressions. When the source relation changes affect more than one materialized view, multiple maintenance expressions need to be evaluated. The techniques of multiquery optimization [28, 29] can be used to detect “common subexpressions” between maintenance expressions in order to derive an efficient global evaluation plan for these maintenance expressions. The maintenance expressions can be evaluated more efficiently if they can be partially rewritten over views already materialized at the DW: the evaluation of parts of the maintenance expression is avoided since their materializations are present at the DW. Moreover, access of the remote data sources and expensive data transmissions are reduced. Materialized views that are added to the DW for reducing the view maintenance cost are called auxiliary views [25, 36]. Obviously, maintaining the auxiliary views incurs additional maintenance cost. However, if this cost is less than the reduction to the maintenance cost of the initially materialized views, it is worth keeping the auxiliary views in the DW. [25, 17] derive auxiliary views to materialize in order to minimize the view maintenance cost.

Minimization of the operational cost. Minimizing the query evaluation cost and the view maintenance cost are conflicting requirements. Low view maintenance cost can be obtained by replicating source relations at the DW. In this case, though, the query evaluation cost is high since queries need to be computed from the replicas of the source relations. Low query evaluation cost can be obtained by materializing at the DW all the input queries. In this case all the input

queries can be answered by a simple lookup. The view maintenance cost is high, though, since complex maintenance expressions over the source relations need to be computed. The input queries may overlap, that is they may share many common subexpressions. By materializing common subexpressions and other views over the source relations, it is possible, in general, to reduce the view maintenance cost. However, these savings must be balanced against higher query evaluation cost. For this reason, one can choose to minimize a linear combination of the query evaluation and view maintenance cost (*operational cost*). The approaches [11, 35, 2, 39, 36, 34, 37] endeavor to minimize the operational cost.

2.2 System oriented constraints

System oriented constraints are dictated by the restrictions of the system, and are transparent to the users.

Space constraint. Although the degradation of the cost of disk space allows for massive storage of data, we cannot consider that the disk space is unlimited. The space constraint restricts the space occupied by the selected materialized views not to exceed the space allocated to the DW for this end. Space constraints are adopted in many works [14, 11, 34, 36, 37, 7]

View maintenance cost constraint. In many practical cases the factor that refrains us from materializing all the views in the DW is not the space constraint (disk space is cheap anyway) but the view maintenance cost. Usually, DWs are updated periodically, e.g. at nighttime, in a large batch update transaction. Therefore the update window must be sufficiently short so that the DW is available for querying and analysis during daytime. The view maintenance cost constraint states that the total view maintenance cost should be less than a given amount of view maintenance time. [12, 18, 7] consider a view maintenance cost constraint in selecting materialized views.

Self Maintainability constraint. A materialized view is self-maintainable [10] if it can be maintained, for any instance of the source relations over which it is defined, and for all source relation changes, using only these changes, the view definition, and the view materialization. The notion is extended to a set of views in a straightforward manner [15]. By adding auxiliary views to a set of materialized views, we can make the whole view set self-maintainable. There are different reasons for making a view set self-maintainable: (a) The remote source relations need not be contacted in turn for evaluating maintenance expressions during view updating. (b) "Anomalies" due to concurrent changes [42, 43] are eliminated and the view maintenance process is simplified. (c) The materialized views can be maintained efficiently even if the sources are not able to answer queries (e.g. legacy systems), or if they are temporarily unavailable (e.g. in mobile systems). By adding auxiliary views to a set of materialized views, the whole view set can be made self-maintainable. Self-maintainability can be trivially achieved by replicating at the DW all the source relations used in the view definitions. The

self-maintainability constraint requires that the set of materialized views taken together is self-maintainable. The approaches [24, 1, 21, 22, 32] aim at making the DW self-maintainable.

Answerability of the queries from the materialized views constraint. In the Data Warehousing approach to the integration of data from the remote data sources [38], the materialized views, necessary for answering the input queries, must be present at the DW. The answerability of the queries constraint requires the input queries to be answerable from the materialized views. This means that there must be a complete rewriting [19, 6] of the queries, initially defined over the source relations, over the materialized views. Clearly, if this constraint is satisfied, the remote data sources need not be contacted for evaluating queries, and, therefore, expensive data transmissions from the DW to the sources, and conversely, are avoided. Some approaches assume a centralized DW environment where the source relations (or source fact tables and dimension tables in case of OLAP queries) are present at the DW site [14, 11, 39, 12] (and [14, 2, 30] for OLAP queries). In this case the answerability of the queries is trivially guaranteed by the presence of the source relations. The answerability of the queries can also be trivially guaranteed by appropriately defining select-project views on the source relations, and replicating them at the DW [41]. (In this case the self-maintainability of the materialized views is also guaranteed.) [35, 36, 34, 37] do not assume a centralized DW environment or replication of part of the source relations at the DW, and explicitly impose the query answerability constraint in selecting views for materialization.

2.3 User oriented constraints

User oriented constraints express requirements of the users.

Answer data currency constraints. An answer data currency constraint puts an upper bound on the time elapsed between the point in time the answer to a query is returned to the user and the point in time the most recent changes of a source relation that are taken into account in the computation of this answer are read (this time reflects the currency of answer data) [33]. Currency constraints are associated with every source relation in the definition of every input query. The upper bound in an answer data currency constraint (minimal currency required) is set by the users according to their needs. This formalization of data currency constraints allows stating currency constraints at the query level and not at the materialized view level as is the case in other approaches [27, 15]. Therefore, currency constraints can be exploited by DW view selection algorithms where the queries are the input, while the materialized views are the output (and therefore are not available). Furthermore, it allows stating different currency constraints for different relations in the same query.

Query response time constraints. A query response

time constraint states that the time needed to evaluate an input query, using the views materialized at the DW, should not exceed a given bound. The bound for each query is given by the users and reflects their needs for fast answers. For some queries fast answers may be required, while for others the response time may not be predominant.

2.4 The general view selection problem

The general view selection problem can now be stated as follows:

Input:

- A set of source relations $\mathbf{R} = \{R_1, \dots, R_n\}$.
- A set of queries $\mathbf{Q} = \{Q_1, \dots, Q_m\}$ over \mathbf{R} .
- Query frequencies f_{Q_1}, \dots, f_{Q_m} , and source relation change propagation frequencies f_{R_1}, \dots, f_{R_n} .
- A set of views $\mathbf{U} = \{U_1, \dots, U_k\}$ over \mathbf{R} that are initially materialized in the DW.
- System oriented and user oriented constraints.
- A cost model and the necessary statistical information concerning the source relations.

Output:

- A set of views \mathbf{V} to additionally materialize in the DW that satisfies all the constraints, and minimizes the operational cost.

This definition of the view selection problem covers not only the initial design of a DW ($\mathbf{U} = \emptyset$), but also its evolution: the DW can contain a set of views initially materialized in it. The input queries can be the queries that the DW has additionally to satisfy (for instance, because the analysts need extra queries).

Different view selection problems addressed previously can be expressed as a special case of the general view selection problem. For instance, the problem of choosing auxiliary views for a materialized view U in order to minimize the overall view maintenance cost [25] is a special case of the general problem where $\mathbf{Q} = \emptyset$, $\mathbf{U} = \{U\}$, and no constraints are imposed. In this case, since $\mathbf{Q} = \emptyset$, the operational cost is the view maintenance cost. As another example, the problem of selecting views that minimize the operational cost under a space constraint in a centralized environment [11] is a special case of the general problem where $\mathbf{U} = \mathbf{R}$, and the only imposed constraint is the space constraint (the answerability of the queries and the self-maintainability constraints are satisfied in this case trivially).

3. MULTIEXPRESSION AND/OR DAGS AND RELATED NOTIONS

In this section we define multiexpression AND/OR dags and their derivatives: multiquery AND/OR dags, query evaluation dags and change propagation dags. We then show how source relation changes can be propagated to the materialized views using change propagation dags. Finally, we define the notion of redundant materialized views. We adopt a natural extension of the relational algebra operations to bags (multisets). This algebra [31,

32] allows defining queries and views that have the SQL bag semantics, and comprises a generalized projection operator [9, 23] to account for grouping/aggregation operations frequently used in Data Warehousing applications.

3.1 Multiquery AND/OR dags

Expression AND/OR dags is a well known way for compactly representing alternative evaluations of an algebraic expression [26]. Extending this notion to account for multiple expressions yields the multiexpression AND/OR dags that allow, in addition, the representation of common subexpression sharing between the expressions [11, 12]. A particular form of an expression AND/OR dag distinguishes between AND nodes and OR nodes [25], and has been developed initially for performing cost-based query optimization. We use here this form of multiexpression AND/OR dags, extended with marked nodes to account for views materialized at the DW. A similar notion is used in [31, 32, 33].

DEFINITION 3.1. Given a set of expressions \mathbf{E} defined over a set of views (and/or relations) \mathbf{U} , a *multiexpression AND/OR dag* \mathcal{G} for \mathbf{E} over \mathbf{U} is a bipartite dag. The nodes of \mathcal{G} are partitioned in AND nodes and OR nodes. An AND node is called *operation node* and is labeled by an operator, while an OR node is called *view node* and is labeled by the expression (view) it computes. In the following we may identify nodes with their labels. An operation node has one or two outgoing edges to view nodes, and one incoming edge from a view node. A view node has one or more outgoing edges (if any) to operation nodes, and one or more incoming edges (if any) from an operation node. \mathcal{G} is not necessarily a rooted dag (that is, it does not necessarily have a single root). All the expressions in \mathbf{E} appear in \mathcal{G} . All the root nodes of \mathcal{G} are view nodes labeled by expressions in \mathbf{E} (but not all the expressions in \mathbf{E} label necessarily root nodes). The sink nodes in \mathcal{G} are labeled by views in \mathbf{U} . View nodes in a multiexpression AND/OR dag *can be marked*. Marked nodes represent materialized views. \square

Sometimes, we may refer to a multiexpression AND/OR dag without mentioning the set of expressions \mathbf{E} and/or the set of views \mathbf{U} .

We also need the notion of a subdag of a multiexpression AND/OR dag:

DEFINITION 3.2. A *subdag* \mathcal{G}' of a multiexpression AND/OR dag \mathcal{G} is a multiexpression AND/OR dag such that:

- (a) Dag \mathcal{G}' is a subdag of dag \mathcal{G} .
- (b) If an operation node of \mathcal{G} is in \mathcal{G}' , all its child view nodes in \mathcal{G} are in \mathcal{G}' .
- (c) The marked nodes in \mathcal{G} that appear as nodes in \mathcal{G}' are the only marked nodes in \mathcal{G}' . \square

A multiexpression dag is a special case of a multiexpression AND/OR dag that provides a single way for evaluating multiple expressions.

DEFINITION 3.3. A *multiexpression dag* for a set of expressions \mathbf{E} over a set of views \mathbf{U} is a multiexpression AND/OR dag for \mathbf{E} over \mathbf{U} such that no view node in it has more than one outgoing edge. \square

We can now define multiquery AND/OR dags.

DEFINITION 3.4. Consider a set of queries \mathbf{Q} defined over a set of source relations \mathbf{R} and a set of materialized views \mathbf{W} over \mathbf{R} . A *multiquery AND/OR dag* \mathcal{G} is an expression AND/OR dag for $\mathbf{Q} \cup \mathbf{W}$ over \mathbf{R} such that:

- (a) All and only the view nodes in \mathbf{W} are marked nodes in \mathcal{G} .
- (b) All and only the sink nodes in \mathcal{G} are labeled by source relations in \mathbf{R} .

The view nodes representing (and labeled by) the queries in \mathbf{Q} are called *query nodes*. Those representing source relations are called *source relation nodes*. A multiquery dag \mathcal{G} is a multiexpression dag for $\mathbf{Q} \cup \mathbf{W}$ over \mathbf{R} that is a subdag of \mathcal{G} . \square

We will not go into the details of multiquery AND/OR dag construction. In the following, we assume that a multiquery AND/OR dag \mathcal{G} for a set of queries \mathbf{Q} is given. A multiquery AND/OR dag constitutes the space from which we draw information on queries, materialized views, source relations, alternative ways of evaluating queries and propagating source relation changes, and subexpression sharing.

3.2 Query evaluation and change propagation dags

Consider a set of queries \mathbf{Q} over a set of source relations \mathbf{R} and a multiquery dag \mathcal{G} . Queries are evaluated using query evaluation dags.

DEFINITION 3.5. A *query evaluation dag in \mathcal{G} for a query $Q \in \mathbf{Q}$* is a multiexpression dag \mathcal{G}_Q , subdag of \mathcal{G} , such that:

- (a) \mathcal{G}_Q is rooted at Q .
- (b) All and only the sink nodes of \mathcal{G}_Q are marked nodes or source relation nodes. \square

Source view changes are propagated to the materialized views using change propagations dags.

DEFINITION 3.6. A *change propagation dag in \mathcal{G} for a source view node $R \in \mathbf{R}$* is a multiexpression dag \mathcal{G}_R , subdag of \mathcal{G} such that:

- (a) All the marked view nodes that are ancestor nodes of R in \mathcal{G} are present in \mathcal{G}_R .
- (b) The root nodes of \mathcal{G}_R are marked nodes.
- (c) All the sink nodes in \mathcal{G}_R are marked nodes or source relation nodes of \mathcal{G} , and R is one of them.
- (d) The non-sink marked nodes in \mathcal{G}_R are ancestor nodes of R . \square

Clearly a change propagation dag is a connected graph. Note that the change propagation dags for the same source relation in different multiquery dags can be different. Further, change propagation dags for different source views in the same or different multiquery dags can be identical.

3.3 Source relation change propagation

We consider two types of changes: insertions and deletions. Modifications are modeled by deletions followed by insertions. Using change propagations dags for the source relations in the same multiquery dag, we can propagate source relation changes to the materialized views that are affected by these changes. The materialized views that are affected by the changes of a source relation R are those that are ancestor view nodes of R in a change propagation dag. Change propagation dags allow the efficient propagation of the changes: common subexpressions between different maintenance expressions are taken into account in order to avoid their re-computation. Furthermore, materialized views are exploited and are not recomputed from scratch.

The changes are propagated along a change propagation dag for a source relation R bottom-up. The starting point is node R whose changes are transmitted by the corresponding source and therefore are available. The sources can also filter source relation changes that are irrelevant, that is changes that have no effect on the state of the materialized views, independently of the source relation state [3, 20]. The changes of a view node are computed using the change of its child view nodes. In order to avoid wasteful insertions and deletions, as well as wasteful data transmissions, we assume that the bag of tuples to be inserted into view V , ∇V , and the bag of tuples to be deleted from V , ΔV are defined to be strongly minimal [8]: $\nabla V \dot{-} V = \emptyset$ and $\nabla V \min \Delta V = \emptyset$. This means that a tuple occurs in the materialized view at least as many times as in the multiset of tuples to be deleted from the view, and that the multiset of tuples to be deleted from and the multiset of tuples to be inserted into a view do not have any tuple in common. Therefore, we can use the maintenance expressions provided in [8] for the typical algebraic operators, and in [23] for the generalized projection operator in order to compute the changes of a view node using the change of its child view nodes.

The maintenance expressions involve, in general, the pre-update state of the child view nodes (that is their state prior to the application of the changes), the pre-update state of the view node, and the changes of the child view nodes. If a view, involved in an expression, is a materialized view, its pre-update state is available. Otherwise, it is computed recursively by its child view nodes in \mathcal{G}_R . When the changes of a view node W in \mathcal{G}_R are computed, they are applied to it if it is a marked node (materialized view). However, if the pre-update state of W is needed in \mathcal{G}_R , this application is postponed until the changes and the pre-update state (if needed) of the parent view nodes of W are computed.

3.4 Redundant views

When computing the changes of a view node using the changes of its child view node(s) in a change propagation dag, the pre-update state of this view node and/or the pre-update state of its child view node(s) may not be needed.

If a view node is a marked node, its pre-update state is available. The pre-update state of a non-marked view node V is computed from the pre-update state of its child view node(s). If the pre-update state of V is not needed, neither for the computation of the changes of V nor by any of its parent view nodes, V is useless in \mathcal{G}_R .

More generally, given a multiquery dag \mathcal{G} , a view node in \mathcal{G} is redundant if it is not used in any query evaluation dag in \mathcal{G} , it is not an initially materialized view, and it is useless in all the change propagation dags in \mathcal{G} that propagate source relation changes to the initially materialized views, and to the materialized views that are used in the query evaluation dags in \mathcal{G} .

If a materialized view V is redundant in \mathcal{G} it can be made virtual without affecting the query evaluation cost of \mathcal{G} . The view maintenance cost of \mathcal{G} is reduced since no changes need to be applied to V and the computation of the changes of V is possibly avoided. A process for detecting redundant materialized views in a multiquery dag is provided in [31].

4. THE DESIGN GOALS IN TERMS OF EXPRESSION AND/OR DAGS

We show in this section how the design goals can be expressed in terms of expression AND/OR dags. This will allow their use by the design algorithms that operate on expression AND/OR dags. We start by discussing cost model issues.

4.1 Cost model

Consider a set of queries $\mathbf{Q} = \{Q_1, \dots, Q_m\}$ over a set of source relations $\mathbf{R} = \{R_1, \dots, R_n\}$, and a multiquery graph \mathcal{G} .

Let \mathcal{G}_{Q_i} be the query evaluation dag for Q_i , $i = 1, \dots, m$ in \mathcal{G} . $E(\mathcal{G}_{Q_i})$ denotes the cost of evaluating Q_i using \mathcal{G}_{Q_i} . The query evaluation cost of \mathcal{G} is

$$E(\mathcal{G}) = \sum_{i=1}^m f_{Q_i} E(\mathcal{G}_{Q_i}).$$

Let \mathcal{G}_{R_i} be the change propagation dags for Q_i , $i = 1, \dots, n$ in \mathcal{G} . $M(\mathcal{G}_{R_i})$ denotes the cost of propagating the changes of R_i to the materialized views using \mathcal{G}_{R_i} . The view maintenance cost of \mathcal{G} is

$$M(\mathcal{G}) = \sum_{i=1}^m f_{R_i} M(\mathcal{G}_{R_i}).$$

The operational cost of \mathcal{G} is

$$O(\mathcal{G}) = c_1 E(\mathcal{G}) + c_2 M(\mathcal{G}).$$

c_1 and c_2 are constants, set by the DW designer, that indicate the importance of the query evaluation cost vs the view maintenance cost. One of them can be 0 if the corresponding cost is not of interest in a specific design problem. Suppose that \mathbf{SG} is a set of multiquery dags. The operational cost of \mathbf{SG} is

$$O(\mathbf{SG}) = \min_{\mathcal{G} \in \mathbf{SG}} O(\mathcal{G}).$$

Let \mathbf{V} be the set of views additionally materialized in the DW. The storage cost for the materialized views in \mathbf{V} is denoted $S(\mathbf{V})$.

The time needed to transmit the changes of source relation R from the remote source to the DW is denoted t_R . The time period between two consecutive transmissions from the source relation R is

$$T_R = 1/f_R.$$

Our approach is independent of the cost model used to assess $E(\mathcal{G}_{Q_i})$ and $M(\mathcal{G}_{R_i})$. The solution returned though, depends on this model.

4.2 Mapping of the design goals

Consider a multiquery dag \mathcal{G} . Suppose that we evaluate the queries using the query evaluation dags in \mathcal{G} , and that we propagate source relation changes using the change propagation dags in \mathcal{G} .

The space constraint simply states that

$$S(\mathbf{V}) \leq S,$$

where S denotes the space allocated to the DW for materializing additional views.

The view maintenance cost constraint states that

$$M(\mathcal{G}) \leq M,$$

where M is the total amount of view maintenance time allowed, specified by the DW designer.

Let \mathcal{G}_R be the change propagation dag for source relation R in \mathcal{G} . If all the sink nodes of \mathcal{G}_R are marked (materialized views) then, clearly, the propagation of the changes of R can be done using \mathcal{G}_R without accessing the remote source relations. If a sink node in \mathcal{G}_R is a source relation node, access of the corresponding remote source is required during the propagation of the changes of R , unless this sink node is redundant. Therefore, the self-maintainability constraint requires that *in each change propagation dag a sink nodes is marked node or a redundant source relation node*.

A query can be evaluated without contacting the remote source relations only if *the sink nodes of the corresponding query evaluation dag are marked*. This is the requirement of the answerability of the queries constraint for all the query evaluation dags in \mathcal{G} .

For answer data currency constraints, the users specify a minimal currency required C_R^Q , for every query Q , and every source relation R over which Q is defined. The data currency constraint concerning query Q and source relation R is satisfied if and only if

$$T_R + t_R + M(\mathcal{G}_R) + E(\mathcal{G}_Q) \leq C_R^Q.$$

Here, for simplicity, we assume that the changes are not applied to the materialized views in a change propagation dag in a specific order. We also assume that the propagation of the changes of a source relation is not delayed by that of another source relation.

For the query response time constraints the users provide a time bound B_Q for every query Q . Clearly, a query response time constraint is satisfied if and only if

$$E(\mathcal{G}_Q) \leq B_Q.$$

5. THE SEARCH SPACE

In this section we determine the search space, and we discuss algorithms for exploring it.

Having expressed the design goals as conditions over the multiexpression AND/OR dags, we can determine the search space for the general view selection problem. This can be done by an algorithm that takes as input a multiquery AND/OR dag \mathcal{G} containing as marked nodes the initially materialized views (if there are any), and systematically examines all the sets of non-materialized views in \mathcal{G} . For each set, it considers that its views are materialized in \mathcal{G} . Then, the set \mathbf{SG} of multiquery dags in \mathcal{G} that satisfy all the constraints is computed. If \mathbf{SG} is not empty, \mathbf{V} is a point in the search space. The operational cost associated with \mathbf{V} is the operational cost of \mathbf{SG} , that is, the minimal operational cost of the multiquery dags in \mathbf{SG} . A view set associated with the lowest operational cost is a solution to the problem (if a solution exists).

The algorithm can be slightly modified to return also the query evaluation and change propagation dags in the multiquery dag that has the minimal operational cost. Different improvements can be devised. These are out of the scope of this paper, our goal being to show how the points in the search space and the associated operational cost are determined.

Clearly such an algorithm is exponential. In [11], the addressed view selection problem is stated NP-hard. Most of the works on view selection problems avoid exhaustive algorithms. The algorithms adopted fall in two categories: deterministic and randomized. In the first category belong greedy algorithms with performance guarantee [14, 11], 0-1 integer programming algorithms [39], A* algorithms [12], and various other heuristic algorithms (e.g. [25, 2, 30, 36]) In the second category belong genetic algorithms [18, 40]. Both categories of algorithms exploit the particularities of the specific view selection problem and the restrictions of the class of queries considered. Under the assumption of the same class of queries, they are expected to perform equally

well on the general view selection problem, since the additional constraints of this last one reduces the size of the search space.

6. CONCLUSION

The initial design of a DW and its evolution require the selection of views for materialization. This selection can be done with different design goals each time. Different view selection problems have been defined and addressed in the past. We have identified and analyzed these design goals, and we have unified the different problems into a general view selection problem that aims at satisfying all the goals together. In order to solve the general view selection problem we have used an AND/OR dag representation for multiple queries and views. We have mapped the design goals into conditions over the AND/OR dag structure, and we have outlined an algorithm that generates the space of view selections that satisfy the constraints, and returns the optimal solution. Our approach is general in that (a) it considers a broad class of queries, (b) it deals not only with static but also with dynamic issues of the design of a DW, and (c) it encompasses as a special case other view selection problems that can result by not considering some of the design goals of the general problem.

We are currently seeking experimental evaluation of our approach through the use of randomized algorithms that do not exhaustively explore the search space of the problem.

7. REFERENCES

- [1] M. O. Akinde, O. G. Jensen, and M. H. Böhlen. Minimizing Detail Data in Data Warehouses. In *Proc. of the 6th Intl. Conf. on Extending Database Technology*, pages 293–307, 1998.
- [2] E. Baralis, S. Paraboschi, and E. Teniente. Materialized view selection in a multidimensional database. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 156–165, 1997.
- [3] J. A. Blakeley, N. Coburn, and P. A. Larson. Updating derived relations: detecting irrelevant and autonomously computable updates. *ACM Transactions on Database Systems*, 14(3):369–400, 1989.
- [4] J. A. Blakeley, P. Larson, and F. W. Tompa. Efficiently updating materialized views. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 61–71, 1986.
- [5] M. Bouzeghoub and Z. Kedad. A Quality-Based Framework for Physical Data Warehouse Design. In *Proc. of the Intl. Workshop on Design and Management of Data Warehouses*, pages 9/1–12, 2000.
- [6] S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. of the 18th ACM Symp. on Principles of Database Systems*, pages 155–166, 1999.
- [7] M. Golfarelli and S. Rizzi. View Materialization for Nested GPSJ Queries. In *Proc. of the Intl. Workshop on Design and Management of Data Warehouses*, pages 10/1–9, 2000.
- [8] T. Griffin and L. Libkin. Incremental Maintenance of Views with Duplicates. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 328–339, 1995.

- [9] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-Query Processing in Data Warehousing Environments. In *Proc. of the 21st Intl. Conf. on Very Large Data Bases*, pages 358–369, 1995.
- [10] A. Gupta, H. Jagadish, and I. S. Mumick. Data Integration using Self-Maintainable Views. In *Proc. of the 5th Intl. Conf. on Extending Database Technology*, pages 140–144, 1996.
- [11] H. Gupta. Selection of Views to Materialize in a Data Warehouse. In *Proc. of the 6th Intl. Conf. on Database Theory*, pages 98–112, 1997.
- [12] H. Gupta and I. S. Mumick. Selection of Views to Materialize Under a Maintenance Cost Constraint. In *Proc. of the 7th Intl. Conf. on Database Theory*, pages 453–470, 1999.
- [13] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing Data Cubes Efficiently. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 1996.
- [14] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing Data Cubes Efficiently. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, 1996.
- [15] R. Hull and G. Zhou. A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 481–492, 1996.
- [16] M. Jarke, M. A. Jeusfeld, C. Quix, and P. Vassiliadis. Architecture and Quality in Data Warehouses: An Extended Repository Approach. *Information Systems*, 24(3):229–253, 1999.
- [17] W. Labio, D. Quass, and B. Adelberg. Physical Database Design for Data Warehousing. In *Proc. of the 13th Intl. Conf. on Data Engineering*, 1997.
- [18] M. Lee and J. Hammer. Speeding up warehouse physical design using a randomized algorithm. In *Proc. of the Intl. Workshop on Design and Management of Data Warehouses, Germany*, pages 3/1–9, 1999.
- [19] A. Levy, A. O. Mendelson, Y. Sagiv, and D. Srivastava. Answering Queries using Views. In *Proc. of the ACM Symp. on Principles of Database Systems*, pages 95–104, 1995.
- [20] A. Y. Levy and Y. Sagiv. Queries Independent of Updates. In *Proc. of the 19th Intl. Conf. on Very Large Data Bases*, pages 171–181, 1993.
- [21] W. Liang, H. Li, H. Wang, and M. E. Orłowska. Making Multiple Views Self-Maintainable in a Data Warehouse. *Data and Knowledge Engineering*, 30(2):121–134, 1999.
- [22] M. Mohania and Y. Kambayashi. Making Aggregate Views Self-Maintainable. *Data and Knowledge Engineering*, 32(1):87–109, 2000.
- [23] D. Quass. Maintenance Expressions for Views with Aggregation. In *Workshop on Materialized Views: Techniques and Applications*, pages 110–118, 1996.
- [24] D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making Views Self Maintainable for Data Warehousing. In *Proc. of the 4th Intl. Conf. on Parallel and Distributed Information Systems*, pages 158–169, 1996.
- [25] K. A. Ross, D. Srivastava, and S. Sudarshan. Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 447–458, 1996.
- [26] N. Roussopoulos. View Indexing in Relational Databases. *ACM Transactions on Database Systems*, 7(2):258–290, 1982.
- [27] A. Segev and W. Fang. Currency-based updates to distributed materialized views. In *Proc. of the 6th Intl. Conf. on Data Engineering*, pages 512–520, 1990.
- [28] T. K. Sellis. Multiple Query Optimization. *ACM Transactions on Database Systems*, 13(1):23–52, 1988.
- [29] K. Shim, T. K. Sellis, and D. Nau. Improvements on a Heuristic Algorithm for Multiple-Query Optimization. *Data & Knowledge Engineering*, 12:197–222, 1994.
- [30] A. Shukla, P. M. Deshpande, and J. F. Naughton. Materialized View Selection for Multidimensional Datasets. In *Proc. of the 24th Intl. Conf. on Very Large Data Bases*, pages 488–499, 1998.
- [31] D. Theodoratos. Detecting Redundancy in Data Warehouse Evolution. In *Proc. of the 18th Intl. Conf. on Conceptual Modeling, Springer LNCS No 1728*, pages 340–353, 1999.
- [32] D. Theodoratos. Complex View Selection for Data Warehouse Self-Maintainability. To appear in *Proc. of the Intl. Conf. on Cooperative Information Systems, Springer, LNCS*, 2000.
- [33] D. Theodoratos and M. Bouzeghoub. Data Currency Quality Satisfaction in the Design of a Data Warehouse. Accepted at the *International Journal of Cooperative Information Systems, Special Issue on Design and Management of Data Warehouses, World Scientific*.
- [34] D. Theodoratos, S. Ligoudistianos, and T. Sellis. Designing the Global Data Warehouse with SPJ Views. In *Proc. of the 11th Intl. Conf. on Advanced Information Systems Engineering, Springer-Verlag, LNCS No 1626*, pages 180–194, 1999.
- [35] D. Theodoratos and T. Sellis. Data Warehouse Configuration. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 126–135, 1997.
- [36] D. Theodoratos and T. Sellis. Designing Data Warehouses. *Data and Knowledge Engineering*, 31(3):279–301, Oct. 1999.
- [37] D. Theodoratos and T. Sellis. Incremental Design of a Data Warehouse. *Journal of Intelligent Information Systems, Kluwer Academic Publishers*, 15(1):7–27, 2000.
- [38] J. Widom. Research Problems in Data Warehousing. In *Proc. of the 4th Intl. Conf. on Information and Knowledge Management*, pages 25–30, Nov. 1995.
- [39] J. Yang, K. Karlapalem, and Q. Li. Algorithms for Materialized View Design in Data Warehousing Environment. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 136–145, 1997.
- [40] C. Zhang and J. Yang. Genetic Algorithm for Materialized View Selection in Data Warehouse Environments. In *Proc. of the 1st Intl. Conf. on Data Warehousing and Knowledge Discovery, Springer-Verlag, LNCS No 1676*, pages 116–125, 1999.
- [41] G. Zhou, R. Hull, and R. King. Generating Data Integration Mediators that Use Materialization. *Journal of Intelligent Information Systems*, 6(2):199–221, 1996.
- [42] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 316–327, 1995.
- [43] Y. Zhuge, H. Garcia-Molina, and J. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In *Proc. of the 4th Intl. Conf. on Parallel and Distributed Information Systems*, 1996.