D. Theodoratos, T. Sellis

Data Warehouse Configuration

Proc. of the 23rd VLDB Conference (VLDB'97)

Athens, Greece, August 1997

Data Warehouse Configuration

Dimitri Theodoratos [*] Timos Sellis [*]

Department of Electrical and Computer Engineering
Computer Science Division
National Technical University of Athens
Zographou 157 73, Athens, Greece
{dth,timos}@dblab.ece.ntua.gr

Abstract

In the data warehousing approach to the integration of data from multiple information sources, selected information is extracted in advance and stored in a repository. A data warehouse (DW) can therefore be seen as a set of materialized views defined over the sources. When a query is posed, it is evaluated locally, using the materialized views, without accessing the original information sources. The applications using DWs require high query performance. This requirement is in conflict with the need to maintain in the DW updated information. The DW configuration problem is the problem of selecting a set of views to materialize in the DW that answers all the queries of interest while minimizing the total query evaluation and view maintenance cost.

In this paper we provide a theoretical framework for this problem in terms of the relational model. We develop a method for dealing with it by formulating it as a state space optimization problem and then solving it using an exhaustive incremental algorithm as well as a heuristic one. We extend this method by considering the case where auxiliary views are stored in the DW solely for reducing the view maintenance cost.

*Research supported by the European Commission under the ESPRIT Program LTR project No 22469 "DWQ: Foundations of Data Warehouse Quality"

Proceedings of the 23rd VLDB Conference
Athens, Greece, 1997

1 Introduction

A Data Warehouse (DW) is a repository of integrated information available for querying and analysis [32]. Data Warehousing is an in-advance approach to the integration of data from multiple, possibly very large, distributed, heterogeneous databases and other information sources [33]. In this approach:

- Selected information from each source is extracted in advance, translated and filtered as needed, merged with relevant information from other sources and stored in a repository.
- When a query is posed, it is evaluated directly on the repository (DW) without accessing the original information sources.

The information stored at the DW can be used by organizations for decision support. This activity makes heavy use of aggregate queries and is called OLAP (On-line Analytical Processing). Aggregations are much more complex than in the case of OLTP (On-Line Transaction Processing) queries [7, 5, 13, 12]. Thus, the data must be available locally at the DW and large multisource queries are executed over the DW.

A DW can be seen as a repository where views over the data from multiple remote information sources are defined and stored materialized. When changes to the base data occur, they must be propagated to the DW. Different update policies can be applied (e.g. immediate or deferred) depending on the client's needs for currency on different parts of the data stored at the

DW. Incremental updating techniques can be more efficient, especially when changes to the base data affect only a small part of it, as is usually the case. In order to compute the changes to the materialized views from the changes to the base data, in most cases, queries must be issued against the base data. This procedure is more time consuming when the data sources are distributed and the transmission costs are important.

1.1 The problem: Data Warehouse configuration

The need for high query performance (low query processing cost) is in conflict with the need for low view maintenance cost. High query performance can be achieved by storing in the DW the result of all the queries of interest. In this case the maintenance cost of the materialized queries might be prohibitively high. By materializing in the DW an appropriately selected set of views we can keep the total query processing cost and the view maintenance cost (*operational cost*) at an acceptable level.

In this paper we deal with the problem of selecting such a set of views. The solution for a given maintenance policy is a compromise between fully materializing all the queries of interest on the one side and keeping replicas of all the base data needed for answering the queries on the other side. More specifically the *DW configuration problem* is formulated as follows: given a set of queries of interest to be issued against the DW, determine a set of views such that:

1. All the queries can be answered using exclusively this set of views, and

2. The operational cost (i.e. the combination of query processing and view maintenance cost) is minimal

The DW configuration problem is more complicated than other query or view maintenance optimization problems using views for the following reasons:

- When views are kept materialized, in order to minimize the query evaluation [30, 4, 13, 12] or the view maintenance cost [20], queries *possibly* need to be answered *using some* materialized views. In the context of the DW configuration problem *all* the queries need to be answered *using exclusively* materialized views. In other words, there is the following additional constraint: *for every query*, there must be a *complete rewriting* using the materialized views [15].

- The DW operational cost is the combination of the cost of two activities. These activities affect each other when we modify the set of views materialized in the DW: the modification may be beneficial for the query evaluation process while being harmful for the view maintenance process and vice versa.

Even though there has been a lot of work on various aspects of materialized views with respect to DWs, there is little or no theoretical work at all on providing a method for configuring a DW. As a consequence the design of a DW is haphazard and the quality of data is often dubious. A formalization of the problem in [11] neglects the fact that queries need to be answered locally, using solely the materialized views.

1.2 Contribution and outline

In this paper we set up a theoretical basis for the DW configuration problem. We then provide a method for solving it for a certain class of relational queries and views. Based on a representation of views using multiquery graphs we model the problem as a state space search problem. Every state is a multiquery graph of the views that are materialized in the DW plus a complete rewriting of the queries over these views. A transition from one state to another transforms the multiquery graph and rewrites completely the queries over the new view set. We search for states having minimal operational cost using an exhaustive algorithm which is also extended with heuristics for pruning the search space. The solution is constructive. Thus, we provide both a set of views to materialize in a DW and a complete rewriting of all the queries over it that minimizes the operational cost. Further, we extend this method, and we compute states having minimal operational cost in the case where auxiliary views are additionally stored in the DW solely for reducing the view maintenance cost [20]. An extreme solution with this approach is when the set of all the stored views is self-maintainable [18]. Our method is general in that it does not consider that the materialized views and the base data are stored in the same database. Further, it is not dependent on the way the query evaluation and view maintenance cost is computed.

The paper is organized as follows. In Section 2, we briefly review related work in the area. In Section 3, we set up a theoretical basis for the DW configuration problem. The problem is modeled as a state space search problem in Section 4, where a solution based on an exhaustive algorithm is provided and the use of heuristics. A solution that stores auxiliary views for reducing the maintenance cost is treated in Section 5. Section 6 contains concluding remarks and possible extensions. An extended version of this work appears on [29].

2 Related work

The DW configuration problem relates to several overlapping research areas.

Answering queries using materialized views has been studied in [14, 34, 30, 15, 4]. The same issue, in connection to aggregate queries and views, has been studied in [7] while in [5], multiset semantics is additionally considered. The problem of optimizing query evaluation in the presence of materialized views has been studied in [30, 4].

The problem of maintaining materialized views has also been the focus of several efforts. Incremental maintenance algorithms are given in [2] for SPJ-queries, in [17] for arbitrary relational algebra expressions, and in [9, 28] for recursive queries. [9, 6] handle multiset semantics while [6] handles aggregation and [19, 9] handle grouping/aggregation queries. [3] derives production rules to maintain selected SQL views. An overview of maintenance problems and techniques with respect to materialized views can be found in [10]. A DW usually holds copies of part of the data of distributed sources. In [24, 23] algorithms for materialized view maintenance in distributed environments are provided. [35, 36] study the problem of updates to the DW when the sources are not database management systems and there is an absence of centralized control. Query independence of updates issues are addressed in [1, 16] while view self- maintainability issues are addressed in [1, 8, 18].

Design problems using views usually follow the following pattern: select a set of views to materialize in order to optimize query evaluation cost, or view maintenance cost or both, eventually in the presence of some constraint. In [13] the problem is addressed in its query optimization form in the context of aggregations and multidimensional analysis under a space constraint. This work is extended in [12] where greedy algorithms are provided, in the same context, for selecting both views and indexes. Further, it is carried over the context of a more general class of queries in [11]. Selecting SQL views to optimize the cost of maintaining an SQL view is studied in [20]. In [11] a formulation of the DW configuration problem is provided for minimizing query evaluation and view maintenance cost. This formulation is different from ours in that it considers that all the base relations at the sources are available locally for computation and in that it does not constraint queries to be completely rewritten over the materialized views.

Finally, another two related problems are the multiple-query optimization problem [26, 27] the caching problem [25, 21, 22].

3 Formal statement of the problem

In this section, we formally set up the DW configuration problem, in terms of the relational model. We first briefly explain the view maintenance procedures and we present the query and view maintenance model and costs. Then the problem is stated in detail and intuitive approaches for dealing with it are presented.

3.1 View maintenance

In general, when the base relations are updated, the materialized in the DW views must be updated too. Different update scenarios can be envisaged. They depend among others on the types of updates, on the activeness of the data sources, on the update policy, and on the update strategy of the views.

If differentials can be sent to the DW, or if log files are available, or more generally, if the updates since the last consultation of the sources from the DW can be computed, an incremental update strategy can be more efficient [17, 6]. Otherwise, a complete rematerialization of the affected views from the snapshots of the base relations can be performed.

In an incremental strategy, when updates arrive at the DW, they are propagated up to the affected views. This is done by: (a) issuing queries back to the base relations, (b) computing updated views and (c) performing view updates. In a rematerialization strategy, a similar procedure is followed where the queries involve only relation names but no differentials (updates).

When computing updates to the views, by propagating up to the affected views the updates of the base relations performed by a transaction, multiple queries might be issued against the base relations [35, 20]. These queries again might contain equivalent subexpressions or more generally, subexpressions such that the one subsumes the other. In this case, the techniques of multiple query optimization [26, 27] allow the detection of such subexpressions and the development of optimal global evaluation plans.

3.2 Query evaluation and view maintenance cost

The use we make of a DW is determined by the set of queries $\mathbf{Q} = \{Q_1, \ldots, Q_l\}$ we issue against it. These queries are expressed over a set of base relations $\mathbf{R} = \{R_1, \ldots, R_n\}$. Every base relation is kept in a remote source or locally, with the DW. The DW contains a set of materialized views, $\mathbf{V} = \{V_1, \ldots, V_m\}$, over $\mathbf{R}$, such that every query in $\mathbf{Q}$ can be rewritten (completely) over $\mathbf{V}$. Thus, all the queries in $\mathbf{Q}$ can be answered locally at the DW, without accessing the base relations in $\mathbf{R}$. Let Q be a query over $\mathbf{R}$. By Q^V, we denote a rewriting of Q over $\mathbf{V}$. This notation is extended to sets of queries. Thus, we write $\mathbf{Q}^V$, for a set containing the queries in $\mathbf{Q}$, rewritten over $\mathbf{V}$. Given $\mathbf{Q}$, a *DW configuration* $\mathbf{C}$ is a pair $< \mathbf{V}, \mathbf{Q}^V >$. Note that we do not distinguish in the notation between view names, view definitions and view materializations (and often, we use the word 'view' for all of them).

The cost of evaluating a query $Q_i^V \in \mathbf{Q}^V$ over the materialized views $\mathbf{V}$ is denoted by $E(Q_i^V)$. Assessing the cost of different evaluation plans, in order to chose the cheapest one, is a standard technique in the process of query evaluation optimization. Thus, any query optimizer [31] could be used to assess the cost $E(Q_i^V)$ of the cheapest evaluation plan. With every query $Q_i^V \in \mathbf{Q}^V$, we associate a weight f_i^Q, indicating the relative frequency of issuing Q_i and its relative importance, with respect to all the queries in $\mathbf{Q}^V$. The *evaluation cost of* $\mathbf{Q}^V$, $E(\mathbf{Q}^V) = \sum_{i \in [1,l]} f_i^Q E(Q_i^V)$.

We assume a set of transaction types $\mathbf{T} = \{T_1, \ldots, T_q\}$ that can update the data sources. In the case of an incremental updating, as in [20], each transaction type determines the updated base relations in a source, the types of updates (insertions, deletions, modifications) to the base relations and the size of each update to a relation. In the case of a rematerialization strategy, each transaction type determines only the updated relations. Thus, there is only a notification for the base relations that have changed. The cost of maintaining the views affected by a transaction type T_i is denoted by $M(T_i)$. This cost involves the cost of transmitting data (update differentials, query data and answer data), computing updates, and performing updates to the affected views. In a distributed environment, the transmission cost is predominant while in a centralized one, the cost of computing and performing updates primarily determines the maintenance cost of the materialized views.

With every transaction type $T_i \in \mathbf{T}$, we associate a weight f_i^T, indicating the relative frequency of this transaction type and its relative importance, with respect to all the transaction types in $\mathbf{T}$. The *maintenance cost of* $\mathbf{V}$, $M(\mathbf{V}) = \sum_{i \in [1,q]} f_i^T M(T_i)$. Notice that there might be views that are not affected by any transaction.

Our approach is general in that it does not consider the maintenance cost to be solely the cost of the (incremental) computation of the updates to the views. Most of the work on view maintenance is restricted by the assumption that base relations and views are stored in a single database which has control over the system [10]. Here, the cost of performing the updates to the views and the cost of transmitting the data needed for the updating, to and from the (remote) sources, can also be taken into account.

Moreover, our approach and method for dealing with the DW configuration problem is independent of the way query evaluation and view maintenance cost is assessed. The solutions suggested, though, do depend on the specific cost model used.

3.3 The DW configuration problem

The *operational cost,* $T(\mathbf{C})$*, of a DW configuration* $\mathbf{C} =< \mathbf{V}, \mathbf{Q}^V >$ is $T(\mathbf{C}) = E(\mathbf{Q}^V) + cM(\mathbf{V})$. The parameter c indicates the relative importance of the evaluation and maintenance cost.

The *DW configuration problem* can now be stated as follows:
Input
A set of base relations $\mathbf{R} = \{R_1, \ldots, R_n\}$.
A set $\mathbf{Q} = \{Q_1, \ldots, Q_l\}$ of queries over $\mathbf{R}$.
For every query Q_i, its weight f_i^Q.
A set of transaction types $\mathbf{T} = \{T_1, \ldots, T_q\}$ over the base relations $\mathbf{R}$.
For every transaction type T_j, its weight f_j^T.
The functions E for the query evaluation cost and M for the view maintenance cost.
A parameter c.
Output
A set of views $\mathbf{V}$ such that there exists a rewriting of $\mathbf{Q}$ over $\mathbf{V}$, $\mathbf{Q}^V$, such that the DW configuration $\mathbf{C} =< \mathbf{V}, \mathbf{Q}^V >$ has minimal operational cost.

The DW configuration $< \mathbf{Q}, \mathbf{Q}^Q >$ has minimal query evaluation cost. Indeed, in this case all the queries in $\mathbf{Q}$ are materialized in the DW and they can be answered by a simple lookup. On the other hand, the DW configuration $< \mathbf{R}, \mathbf{Q} >$ has minimal update computation cost. In this case replicas of all the base relations appearing in $\mathbf{Q}$ are materialized and they can be maintained without any update computation.

A solution to the problem is a compromise between these two extremes.

4 A state space search based algorithm

We model the DW configuration problem as a state space search problem for a class of relational queries and views. In this section we consider the case where other materialized views are not used during the maintenance process of a view. This assumption is relaxed in the next section where we treat the general case. Thus for now we consider that in a DW configuration $< \mathbf{V}, \mathbf{Q}^V >$ every materialized view in $\mathbf{V}$ appears in the rewriting of the queries in $\mathbf{Q}^V$. We then present an exhaustive incremental algorithm for selecting the state corresponding to a DW configuration having minimal operational cost, and we suggest heuristics.

We consider the class of relational views (queries) of the form $\sigma_F(R_1 \times \ldots \times R_k)$. The formula F is a conjunction of comparisons of the form $x\ op\ y + c$ or $x\ op\ c$ where op is one of the comparison operators $=$, $<$, $\leq$, $>$, $\geq$ (but no $\neq$), c is an integer valued constant and x, y are attribute names. Conjuncts involving attributes from only one relation are called *selection predicates*, while conjuncts involving attributes from two relations are called *join predicates*. Attributes of every R_i are involved with those of at least one other R_j in a predicate join in F. All the R_is are distinct.

A formula involving unequalities ($\neq$), disjunction and negation can be handled by replacing unequalities by disjunctions of two strict inequalities, converting it into disjunctive normal form, and eliminating negations. Then each disjunct can be considered separately (though this conversion may cause the number of comparisons to grow exponentially). Thus, in the following we consider F to be a conjunction of comparisons as above.

4.1 The states

A set of views $\mathbf{V}$ can be represented by a *multiquery graph*. A multiquery graph allows the compact representation of multiple views by merging the query graphs of each view. For a set of views $\mathbf{V}$, the corresponding multiquery graph, $\mathbf{G}^V$, is a multigraph defined as follows:

1. The set of nodes of $\mathbf{G}^V$ is the set of relations appearing in the views $V \in \mathbf{V}$.

2. For every join predicate F_p in a view $V = \sigma_F(R_1 \times \ldots \times R_k)$ involving attributes of the relations R_i and R_j there is an edge between R_i and R_j labeled as $V : F_p$. Such an edge is called *join edge*.

3. For every selection predicate F_p in a view V involving attributes of the relation R_i, there is an edge from R_i to itself labeled as $V : F_p$. For every view $V = R_i$ there is an edge from R_i to itself labeled as $V : T$. The symbol T denotes here a valid formula. Both these edges are called *selection edges*.

The query graph of a view V can be seen as the multiquery graph of the view set $\{V\}$ where the view name V does not appear in the edge labels.

Example 4.1 Suppose that a view set is $\mathbf{V} = \{V_1, V_2\}$, where the views V_1 and V_2 (in a form using joins) are as follows: $V_1 = R \bowtie_{F_1^1} \sigma_{C_1^1}(S) \bowtie_{F_2^1} \sigma_{C_2^1}(T)$ and $V_2 = S \bowtie_{F_1^2} \sigma_{C_1^2}(T) \bowtie_{F_2^2} \sigma_{C_2^2}(U)$. The base relation schemas are: $R(A, B)$, $S(C, D, E)$, $T(G, K, H)$ and $U(M, L)$. The selection and join conditions are given below:
$C_1^1 : E < 3$, $C_2^1 : K < H{+}3$, $C_1^2 : K < H$, $C_2^2 : L > 5$,
$F_1^1 : A < C \wedge B = D$, $F_2^1 : C = G$,
$F_1^2 : C \leq G \wedge C \geq K$, $F_2^2 : G \leq M$.
The corresponding multiquery graph $\mathbf{G}^V$ is depicted in Figure 1. The order of appearance of the attributes in the join edge labels indicates also which relation they belong to. □

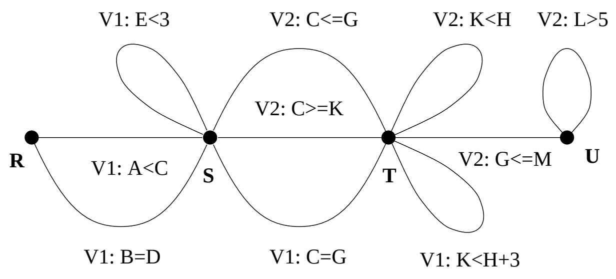

Figure 1: A multiquery graph

A multiquery graph contains all the information about the views in $\mathbf{V}$. Thus, in the following we may use this notation for sets of views. In dealing with the configuration of DWs, besides selecting the set of views $\mathbf{V}$ to materialize in the DW, we are also interested in finding a complete rewriting of $\mathbf{Q}$ over $\mathbf{V}$, $\mathbf{Q}^V$, (i.e. a DW configuration $\mathbf{C} =< \mathbf{V}, \mathbf{Q}^V >$) such that $\mathbf{C}$ has minimal operational cost. In the rest of the paper we address the DW configuration problem as it is stated in the previous section where the output is a DW configuration $\mathbf{C} =< \mathbf{V}, \mathbf{Q}^V >$, and not only a view set $\mathbf{V}$.

States are DW configurations $< \mathbf{G}^V, \mathbf{Q}^V >$. Thus in a state $< \mathbf{G}^V, \mathbf{Q}^V >$ every view in $\mathbf{G}^V$ appears in a query rewriting in $\mathbf{Q}^V$.

4.2 The transitions

In order to define transitions between states we introduce the following three *state transformation rules* that can be applied to a state $s =< \mathbf{G}^V, \mathbf{Q}^V >$. Each state transformation rule consists of a transformation rule of the multiquery graph $\mathbf{G}^V$ and a rewrite rule of the set of queries $\mathbf{Q}^V$.

Selection edge cut:
If e is a selection edge in $\mathbf{G}^V$ of a node R labeled as $V : F_p$, construct a new multiquery graph as follows: (a) if e is the unique selection edge of R labeled by V, then replace its label by $V_1 : T$, where V_1 is a new view name. (b) otherwise, remove e from $\mathbf{G}^V$ and replace every occurrence of V in $\mathbf{G}^V$ by a new view name V_1. New view names should not already appear in $\mathbf{G}^V$. Replace any occurrence of V in $\mathbf{Q}^V$, by the expression $\sigma_{F_p}(V_1)$. Note that in case (a) above, V_1 is the base relation R.

Join edge cut:
If e is a join edge labeled as $V : F_p$ in $\mathbf{G}^V$, remove e from $\mathbf{G}^V$ and construct a new multiquery graph, as follows: (a) if the removal of e does not divide the query graph of V in $\mathbf{G}^V$ into two disconnected components, then replace every occurrence of V in $\mathbf{G}^V$, by a new view name V_1. (b) otherwise, replace every occurrence of V in $\mathbf{G}^V$ by a new view name V_1 in the one component and by a new view name V_2 in the other component. If a component (say the first) is a single node without edges, then add in $\mathbf{G}^V$ a selection edge to this node labeled as $V_1 : T$. In this case V_1 is a base relation. Do similarly for the other component. Note that the removal of edge e may not divide the query graph of V in $\mathbf{G}^V$ into two disconnected components because there are also other edges between the same nodes (labeled by V of course) or because the query graph of view V is cyclic and e is part of a cycle.
Replace any occurrence of V in $\mathbf{Q}^V$, in case (a) above, by the expression $\sigma_{F_p}(V_1)$ and in case (b), by the expression $\sigma_{F_p}(V_1 \times V_2)$.

View merging:
If the query graphs of two views V and V_1 in $\mathbf{G}^V$ have the same set of nodes and each predicate in their query graphs is either implied by a predicate of the other view or implies a predicate of the other view, then construct a new multiquery graph as follows:
Remove from $\mathbf{G}^V$ each edge labeled by a predicate of one of the views that imply a predicate of the other view and is not implied by a predicate of the first view. Replace any occurrence of V and V_1 in $\mathbf{G}^V$ by a new view name V_2. This replacement results in one edge out of two every time an edge of V and an edge of V_1 are labeled by the same predicate. A predicate p implies a predicate p' if p is more restrictive than p'. Clearly the implication entails that both predicates involve the same attributes.
Replace any occurrence of V in $\mathbf{Q}^V$ by the expression $\sigma_{F_p}(V_1)$, where F is the conjunction of the predicates p in V such that: p implies a predicate in V_1 and is not implied by a predicate in V. Do similarly for V_1.
A special case appears when each predicate of view V_1 is implied by a predicate of view V (and each predicate of view V implies or is implied by a predicate of view V_1). In this case, remove all the edges labeled by V from $\mathbf{G}^V$. Replace only any occurrence of V in $\mathbf{Q}^V$ by the expression $\sigma_{F_p}(V_1)$ as previously. If additionally, each predicate of view V is implied by a predicate of view V_1, replace only any occurrence of V in $\mathbf{Q}^V$ by V_1.

Example 4.2 Consider the multiquery graph $\mathbf{G}^V$ of the example 4.1 and the query set $\mathbf{Q}^V = \{Q_1, Q_2\}$, where, $Q_1 = V_1$ and $Q_2 = V_2$. We apply subsequently different state transformation rules. New view names may be introduced in the multiquery graph and in the rewritings of the query definitions during the application of these rules.

By applying the join edge cut rule to the join edge labeled as $V_1 : A < C$, we obtain the multiquery graph depicted in Figure 2. The query Q_1 is rewritten as follows: $Q_1 = \sigma_{A<C}(V_3)$. The query $Q2$ is not affected by this transformation. In Figure 3 is depicted the multiquery graph resulting after the application of the join edge cut rule to the join edges labeled as $V_3 : B = D$ and $V_2 : G \leq M$. The queries Q_1 and Q_2 are now rewritten as follows: $Q_1 = V_4 \bowtie_{F_1^1} V_5$ and $Q_2 = V_6 \bowtie_{F_2^2} V_7$. Note that $V_4 = R$. In Figure 4, the selection edge cut rule has been applied to the selection edge labeled as $V_5 : E < 3$ and the join edge cut rule has been applied to the join edge labeled as $V_6 : C \geq K$ of the multiquery graph depicted in Figure 3. The queries Q_1 and Q_2 are rewritten as follows: $Q_1 = V_4 \bowtie_{F_1^1} \sigma_{C_1^1}(V_8)$ and $Q_2 = \sigma_{C \geq K}(V_9) \bowtie_{F_2^2} V_7$.

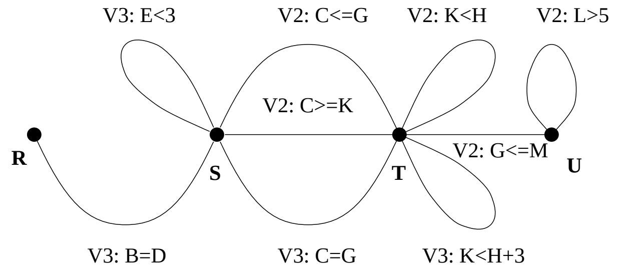

Figure 2: The multiquery graph after an application of the join edge cut rule

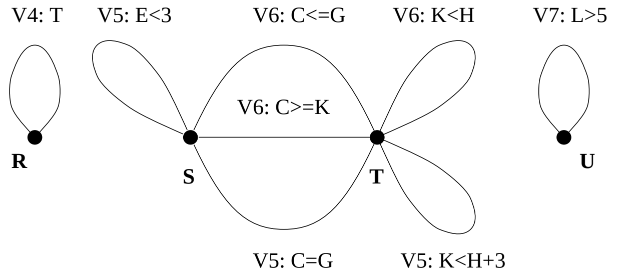

Figure 3: The multiquery graph after applications of the join edge cut rule

The views V_8 and V_9 are defined over the same set of nodes and the join predicate $C = G$ of V_8 is more restrictive than the join predicate $C \leq G$ of V_9 while the selection predicate $K < H$ of V_9 is more restrictive than the selection predicate $K < H + 3$ of V_8. Thus we can apply the view merging rule on the views V_8

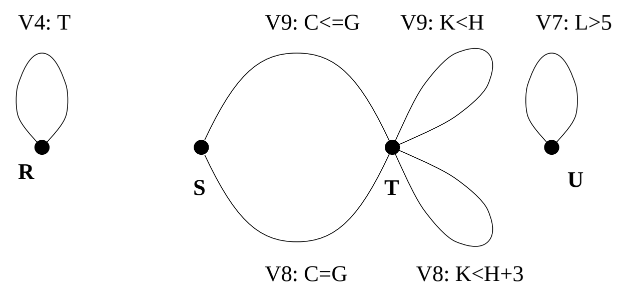

Figure 4: The multiquery graph after applications of edge cut rules

and V_9. The multiquery graph in Figure 5 results after the application of this rule. The final rewriting of the queries is as follows: $Q_1 = V_4 \bowtie_{F_1^1} \sigma_{F_2^1 \wedge C_1^1}(V_{10})$ and $Q_2 = \sigma_{C_1^2 \wedge C \geq K}(V_{10}) \bowtie_{F_2^2} V_7$. □

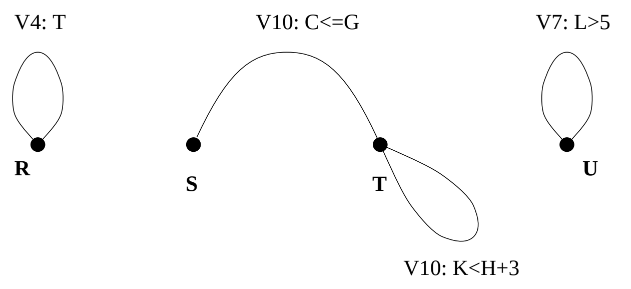

Figure 5: The multiquery graph after an application of the view merging rule

Proposition 4.1 *Consider a state $s =< \mathbf{G}^V, \mathbf{Q}^V >$, where $\mathbf{V}$ is a set of views over a set of base relations $\mathbf{R}$. By applying any of the three state transformation rules to s we obtain the multiquery graph $\mathbf{G}^{V'}$ of a set of views $\mathbf{V}'$ over $\mathbf{R}$ and a complete rewriting of $\mathbf{Q}$ over $\mathbf{V}'$, i.e. a new state $< \mathbf{G}^{V'}, \mathbf{Q}^{V'} >$.* □

Proof: The proof is obvious. □

The previous proposition stands as a soundness statement for the state transformation rules. In order to provide a completeness result we need the notion of the *full form* of a query [29]. When a query is in full form, if its set of predicates $\mathbf{F}$ implies a predicate p, there is a predicate in $\mathbf{F}$ that implies p.

Theorem 4.1 *Let $\mathbf{Q}$ be a set of queries in full form and $\mathbf{C} =< \mathbf{V}, \mathbf{Q}^V >$ be a DW configuration such that every V in $\mathbf{V}$ appears in $\mathbf{Q}^V$. Then there is a state $s' =< \mathbf{G}^{V'}, \mathbf{Q}^{V'} >$ obtained by applying subsequently a finite sequence of state transformation rules on the state $s_0 =< \mathbf{G}^Q, \mathbf{Q}^Q >$ such that:*
(a) There is a 1:1 mapping f from $\mathbf{V}$ onto $\mathbf{V}'$ such that $\forall V \in \mathbf{V}$, V contains $f(V)$.
(b) For every query $Q^V \in \mathbf{Q}^V$, the query $Q^{V'}$ involves exactly the images of the views in Q^V with respect to f. □

Let $\mathbf{S}$ be the set of all the states. There is a *transition* $T(s, s')$ from state s to state s' iff s' can be obtained by applying any of the three state transformation rules to s. With every state $s =< \mathbf{G}^V, \mathbf{Q}^V >$, a *cost* is associated through the function $cost : \mathbf{S} \rightarrow \mathbb{R}$. This is the operational cost of the DW configuration $< \mathbf{V}, \mathbf{Q}^V >$.

Assume now a monotone cost model for evaluating queries and maintaining views. A cost model is *monotone* if the cost of computing a query Q defined over the views $V_1, \ldots, V_n$ is not greater than the cost of computing a query Q', contained by Q, and defined over the views $V_1', \ldots, V_n'$, when V_i contains V_i', $i \in [1, n]$. As a consequence of theorem 4.1, if the views in the DW are maintained by issuing queries back to the base relations and multiquery optimization is not performed on these queries, one is guaranteed to find a state corresponding to a DW configuration having minimal operational cost, by applying the above transformations.

4.3 An exhaustive incremental algorithm

We present now an exhaustive algorithm for the DW configuration problem. The basic outline of the algorithm is shown in Figure 6. The algorithm considers as an initial state the state $s_0 =< \mathbf{G}^Q, \mathbf{Q}^Q >$. It then produces all the subsequent states (expands the state s). New states are expanded in their turn until no more states are left unexpended. States are supposed to be kept along with their cost. Two states are considered here to be identical if the one can be obtained from the other by renaming the views and by reordering views in the cartesian product and conjuncts in the selection formula of the rewritten queries. Clearly the algorithm terminates and returns a state having minimal operational cost.

The cost of a new state can be computed *incrementally* as follows: a state transformation rule modifies one view or merges two views in $\mathbf{G}^V$ defined over the same set of base relations and produces one or two views. Then it rewrites some queries from $\mathbf{Q}^V$. Usually, there is only a small subset of the queries in $\mathbf{Q}^V$ that are affected by the transformation. These are the queries that are defined over the affected, by the transformation, view(s). Also, a view in $\mathbf{G}^V$ is usually affected only by a small subset of the transaction types. These are the transaction types that modify a base relation appearing in its definition. In this section we consider that views are maintained by issuing queries back to the base relations if necessary and that other views are not used in the maintenance process of a view. Let $T(s, s')$ be a transition from state s to state s'. The cost of a state $s =< \mathbf{G}^V, \mathbf{Q}^V >$ is the composition of the evaluation cost , E, of the queries in

```
begin
   compute $cost(s_0)$;
   $open = \{s_0\}$;  $closed = \emptyset$;
   while ($open \neq \emptyset$)
      consider a state $s$ in $open$;
      for every transition $T(s,s')$
         if $s' \notin open \cup closed$ then
            (incrementally) compute $cost(s')$
            $open = open \cup \{s'\}$
         endif
      endfor
      $open = open - \{s\}$;  $closed = closed \cup \{s\}$
   endwhile;
   return the state $s \in closed$ having the
   minimal $cost(s)$
end.
```

Figure 6: An exhaustive algorithm

$\mathbf{Q}^V$ and the maintenance cost, M, of the views in $\mathbf{G}^V$. The increment ΔE to the query evaluation cost from the state s to the state s' can be determined by computing only the weighted sum of the evaluation cost of the affected queries in the states s and s' and taking their difference. The increment ΔM to the view maintenance cost from the state s to the state s' can be determined as follows: let $T_1, \ldots, T_q$ be the transaction types that modify a base relation appearing in the definition of the view(s) affected by the transformation. Compute the costs $\sum_{i\in[1,q]} f_i^T M(T_i)$ in the states s and s' and take their difference. These costs represent the cost of propagating the updates of the transaction types in $\mathbf{T}$ to the views of the states s and s'. If multiquery optimization is not performed on the queries issued for updating the views, the computation of ΔM can be further simplified to the following procedure:
(a) compute the weighted sum of the maintenance cost of the view(s) affected by the transformation for every transaction type that affects it (them) in the state s.
(b) compute the weighted sum of the maintenance cost of the view(s) produced by the transformation for every transaction type that affects it (them) in the state s'.
(c) take the difference of (b) and (a).

An exhaustive algorithm as the previous one can be very expensive for a big number of complex queries. This should not be a problem since the configuration of a DW is not supposed to be done very frequently. Nevertheless, heuristics that can prune the search space are discussed below.

4.4 Heuristic pruning of the search space

The algorithms presented here start with a state having minimal query evaluation cost and follow transitions that do not reduce this cost. The view maintenance cost is not expected to increase when the view merging transformation rule is applied and it is surely reduced when the view merging rule transformation eliminates one view. The reduction of the maintenance cost from the application of the selection edge cut rule or the application of the join edge cut rule when it does not generate two separate subviews, if any, is not important. Their contribution to the reduction of the maintenance cost might be more visible later if their application allows the later application of the view merging rule. Moreover, the application of the join edge cut rule that does not generate two separate subviews is necessary for the application of the join edge cut rule that generates two separate subviews.

Favoring the view merging transformation. The previous remarks suggest for the following heuristic rule application restrictions:
(a) Do not apply the selection edge cut rule on a selection edge labeled by a view V if there is no other view in $\mathbf{G}^V$ defined exactly over the same nodes as V.
(b) Do not apply the join edge cut rule on a join edge between two nodes labeled by a view V, if there is no other view in $\mathbf{G}^V$ defined exactly over the same nodes as V. In this case, a new state transformation rule can be used: the *multiple join edge cut rule.* This rule removes all the edges labeled by V (there may be only one) between these nodes and divides the query graph of V in $\mathbf{G}^V$ in two components. Its effect is the same as the repeated application (in any order) of the join edge cut rule on the edges labeled by V between these nodes, until all these edges are removed.

The previous heuristic restrictions allow the faster reach of states where the view merging rule can be applied, by avoiding intermediate states.

Using only the multiple join edge cut rule. In distributed environments, the maintenance cost depends largely on the transmission costs. In this case removing some of the connections that exist between base relations through joins might be decisive in the reduction of the maintenance cost. The need to query the base relations of a remote data source, when a materialized view must be updated, can be reduced or eliminated. This setting can be treated heuristically by discarding the selection and join edge cut rules and using the multiple join edge cut rule instead. Of course this treatment will be done at the expense of the possibility of applying the view merging rule.

5 Materializing more views for reducing the operational cost

Keeping auxiliary materialized views in a database can be used to reduce the evaluation cost of queries [25, 21, 4]. But, keeping auxiliary materialized views in a DW can be used as well to reduce the view maintenance cost. Indeed, we have already mentioned that in general, when a materialized view is updated, in response to an update to a base relation, queries must be issued back to the base relations and the updated view must be computed from the answers. Now, if some materialized views are kept ad hoc in the DW, then (a) all or some of these answers can be obtained locally, without accessing the (remote) base relations and (b) some of the computations can be avoided.

Example 5.1 Suppose that we have to incrementally update the materialized view $V = R \bowtie_{F_1} \sigma_C(S) \bowtie_{F_2} T$ in response to the insertion of the tuples in ΔT to the base relation T. Suppose that ΔT is available at the DW. Then the tuples defined by the incremental expression $\Delta V = R \bowtie_{F_1} \sigma_C(S) \bowtie_{F_2} \Delta T$ must be inserted into V [17]. If the view $\sigma_C(S)$ is additionally materialized in the DW, there is no need to issue the query $\sigma_C(S)$ against the base relation S in order to compute ΔV. Even better, if the view $R \bowtie_{F_1} S$ is kept materialized, ΔV can be computed locally. In that case, there is no need to access the base relations R and S at all. The set of views V and $R \bowtie_{F_1} S$ is self maintainable with respect to insertions into T. The precomputed view $R \bowtie_{F_1} S$ can be used with ΔT to compute the insertions ΔV to V, while $R \bowtie_{F_1} S$ is not affected by the change. □

Obviously, there is a cost associated with the process of maintaining the auxiliary materialized views. But, 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.

A set of views is self-maintainable if it can be maintained using only the content of the views and the base relations changes i.e. without accessing the base relations [18]. Even if the set of materialized views in a DW is not self-maintainable, adding some properly selected materialized views in the DW may reduce the overall need for accessing the base relations when maintaining the DW. Moreover, their presence in the DW can reduce the cost of computing the updates to the views [20]. Thus the overall maintenance cost may be reduced.

The module of the system that is responsible for the view maintenance should be able to take advantage of the presence of the auxiliary materialized views in the DW. These views are not used for answering the queries in **Q**. They are useful solely for reducing the overall view maintenance cost. Since they do not affect the query evaluation process, they also reduce the operational cost of the DW.

In [29] we present in detail the modified algorithms where auxiliary views are also taken into account.

6 Conclusion and possible extensions

The DW configuration problem is the problem of selecting a set of views to materialize in the DW that answers all the queries of interest while minimizing the combination of the query evaluation and view maintenance cost. We set up a theoretical basis for this problem in terms of the relational model. It is then formulated as a state space optimization problem in two cases: (a) when only views necessary for answering the queries of interest are materialized, and (b) when auxiliary views can be additionally materialized with the purpose to be used in reducing the overall maintenance cost. The formulation of the problem allows both: to compute the set of views and to find a complete rewriting of the queries over it. We have designed an exhaustive algorithm and we have provided heuristics for pruning the search space in different cases.

Extensions of this work include considering a broader class of queries. In particular projections can be handled in the same context by keeping on each node of the multiquery graph the attributes that are projected out for every view defined over this node. The set of state transformation rules presented here can be extended in order to capture the new features introduced by the projection.

Access structures have not been considered in this work. Access structures though can be useful both for evaluating queries and maintaining materialized views. Their presence in the DW incurs additional maintenance cost. Thus, adding them in the DW influences the choice of views to materialize. Selecting a set of access structures and views to store in a DW in order to minimize the operational cost is also a subject of big interest.

[18] derives, using key and referential integrity constraints, a set of auxiliary views for a given view such that the given and the auxiliary views together are self-maintainable. The present work can be extended by developing methods for taking into consideration integrity constraints in the DW configuration process.

Finally, in this work we consider that we have no space restrictions in the DW. Actual DW systems use secondary storage space of the size of Terabytes. Thus we can reasonably consider that for a plethora of applications the space is not a problem. However, solving the DW configuration problem under space restrictions is also a topic of our current research work.

References

[1] J. A. Blakeley, N. Coburn, and P. A. Larson. Updating derived relations: detecting irrelevant and autonomously computable updates. *ACM TODS*, 14(3):369–400, 1989.

[2] J. A. Blakeley, P. Larson, and F. W. Tompa. Efficiently updating materialized views. In *Proc. ACM SIGMOD*, pages 61–71, 1986.

[3] S. Ceri and J. Widom. Deriving production rules for incremental view maintenance. In *Proc. of VLDB*, pages 577–589, 1991.

[4] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing Queries with Materialized Views. In *Proc. ICDE*, pages 190–200, 1995.

[5] S. Dar, H. V. Jagadish, A. Y. Levy, and D. Srivastava. Answering SQL Queries with Aggregation using Views. In *Proc. of VLDB*, pages 318–329, 1996.

[6] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *Proc. ACM SIGMOD*, pages 328–339, 1995.

[7] A. Gupta, V. Harinarayan, and D. Quass. Aggregate-query processing in data warehousing environments. In *Proc. of VLDB*, pages 358–369, 1995.

[8] A. Gupta, H. Jagadish, and I. S. Mumick. Data Integration using Self-Maintainable Views. In *Proc. EDBT*, pages 140–144, 1996.

[9] A. Gupta, I. Mumick, and V. Subrahmanian. Maintaining views incrementally. In *Proc. ACM SIGMOD*, pages 157–166, 1993.

[10] A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, technics and applications. *Data Engineering*, 18(2):3–18, 1995.

[11] H. Gupta. Selection of Views to Materialize in a Data Warehouse. In *Proc. ICDT*, pages 98–112, 1997.

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

[13] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing Data Cubes Efficiently. In *Proc. ACM SIGMOD*, 1996.

[14] P.-A. Larson and H. Yang. Computing Queries from Derived Relations. In *Proc. of VLDB*, pages 259–269, 1985.

[15] A. Levy, A. O. Mendelson, Y. Sagiv, and D. Srivastava. Answering Queries using Views. In *Proc. ACM PODS*, pages 95–104, 1995.

[16] A. Y. Levy and Y. Sagiv. Queries Independent of Updates. In *Proc. of VLDB*, pages 171–181, 1993.

[17] X. Qian and G. Wiederhold. Incremental Recomputation of Active Relational Expressions. *IEEE TKDE*, 3(3):439–450, 1991.

[18] D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making Views Self Maintainable for Data Warehousing. In *Proc. PDIS*, 1996.

[19] R. Ramakrishnan, K. Ross, D. Srivastava, and S. Sudarshan. Efficient incremental evaluation of queries with aggregation. In *International Logic Programming Sumposium*, pages 204–218, 1994.

[20] K. A. Ross, D. Srivastava, and S. Sudarshan. Materialized View Maintenance and Integrity Constraint Checking: Trading Space for Time. In *Proc. ACM SIGMOD*, pages 447–458, 1996.

[21] N. Roussopoulos. The Incremental Access Method of View Cache: Concepts Algorithms and Cost Analysis. *ACM TODS*, 16(3):535–563, 1991.

[22] P. Scheurmann, J. Shim, and R. Vingralek. WATCHMAN: A Data Warehouse Intelligent Cache Manager. In *Proc. of VLDB*, pages 51–62, 1996.

[23] A. Segev and W. Fang. Currency-based updates to distributed materialized views. In *Proc. ICDE*, pages 512–520, 1990.

[24] A. Segev and J. Park. Updating distributed materialized views. *IEEE TKDE*, 1(2):173–184, 1989.

[25] T. K. Sellis. Intelligent caching and indexing techniques for relational database systems. *Information Systems*, 13(2):175–185, 1988.

[26] T. K. Sellis. Multiple Query Optimization. *ACM TODS*, 13(1):23–52, 1988.

[27] K. Shim, T. K. Sellis, and D. Nau. Improvements on a heuristic algorithm for multiple-query optimization. *Data & Knowledge Engineering*, 12:197–222, 1994.

[28] M. Staudt and M. Jarke. Incremental Maintenance of Externally Materialized Views. In *Proc. of VLDB*, pages 75–86, 1996.

[29] D. Theodoratos and T. Sellis. Configuring Data Warehouses. *Technical Report, Knowledge and Data Base Systems Lab., Electrical and Computer Eng. Dept., NTU Athens*, June 1997.

[30] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for physical data independence. In *Proc. of VLDB*, pages 367–378, 1994.

[31] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 2. Computer Science Press, 1989.

[32] J. Widom, editor. *Data Engineering, Special Issue on Materialized Views and Data Warehousing*, volume 18(2). IEEE, 1995.

[33] J. Widom. Research problems in data warehousing. In *Proc. CIKM*, pages 25–30, Nov. 1995.

[34] H. Yang and P.-A. Larson. Query Transformation for PSJ-queries. In *Proc. of VLDB*, pages 245–254, 1987.

[35] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View Maintenance in a Warehousing Environment. In *Proc. ACM SIGMOD*, pages 316–327, 1995.

[36] Y. Zhuge, H. Garcia-Molina, and J. Wiener. The Strobe Algorithms for Multi-Source Warehouse Consistency. In *Proc. PDIS*, 1996.