Designing the Global Data Warehouse with SPJ Views *

Dimitri Theodoratos Spyros Ligoudistianos Timos Sellis

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

Abstract. A global Data warehouse (DW) integrates data from multiple distributed heterogeneous databases and other information sources. A global DW can be abstractly seen as a set of materialized views. The selection of views for materialization in a DW is an important decision in the implementation of a DW. Current commercial products do not provide tools for automatic DW design.
In this paper we provide a generic method that, given a set of SPJ-queries to be satisfied by the DW, generates all the 'significant' sets of materialized views that satisfy all the input queries. This process is complex since 'common subexpressions' between the queries need to be detected and exploited. Our method is then applied to solve the problem of selecting such a materialized view set that fits in the space allocated to the DW for materialization and minimizes the combined overall query evaluation and view maintenance cost. We design algorithms which are implemented and we report on their experimental evaluation.

1 Introduction

In the Data Warehousing approach to the integration of data from multiple distributed heterogeneous databases and other information sources, data are extracted in advance from the sources and stored in a repository (Data Warehouse - DW). When a query is issued against the DW, it is evaluated locally, without accessing the original information sources. DWs are used by companies for decision support applications. Therefore, ensuring high query performance is one of the most significant challenges when implementing a DW.

A DW can be abstractly seen as a set of materialized views defined over source relations. When the source relations change, the materialized views need to be updated. Different maintenance policies (deferred or immediate) and maintenance strategies (incremental or rematerialization) can be applied [2]. This choice depends on the needs of the specific application for currency of the data and view maintenance performance.

* Research supported by the European Commission under the ESPRIT Program LTR project "DWQ: Foundations of Data Warehouse Quality"

A typical DW architecture [1] comprises tree layers. The data at each layer is derived from the data of lower layers. At the lowest layer there are the distributed operational data sources. The central layer is the *global* or *principal Data Warehouse*. The upper layer contains the *local DWs* or *Data marts*. The global DW integrates the data from the distributed data sources. Data marts contain highly aggregated data for extensive analytical processing. They are also probably less frequently updated than global DWs. Here we deal with global DWs. In the following, DW refers to the global DW.

In this paper we address the issue of selecting views for materialization in a DW given a set of input queries. This is an important decision in the implementation of a DW. Current commercial products do not provide tools for automatic DW design.

The materialized view selection issue is complex. One of the reasons of its complexity lies on the fact that the input queries may contain subexpressions that are identical, equivalent or more generally subexpressions such that one can be computed from the other. We describe these subexpressions by the generic term *common subexpressions*. Common subexpressions in the input queries need to be detected and exploited in the view selection process. They can significantly reduce the view maintenance cost and the space needed for materialization.

Computing the input queries using exclusively the materialized views requires the existence of a complete rewriting of the queries over the materialized views [7]. Many formulations of the problem of selecting views to materialize in a DW [3, 14], do not explicitly impose this condition.

Contribution. In this paper we present a generic method for designing a global DW. This method detects and exploits common subexpressions between the queries and guarantees the existence of a complete rewriting of the queries over the selected views. We provide a set of transformation rules for Select-Project-Join (SPJ) queries that, starting with the input queries, generate alternative view selections for materialization and a complete rewriting of the queries over the selected views. These transformation rules are sound, minimal and complete; in this sense all the "significant" view selections are non-redundantly generated.

We then address the following problem (called *DW design problem*):
Given a set of queries to be satisfied by the DW, select a set of views to materialize in the DW such that:

1. The materialized views fit in the space available at the DW.
2. All the queries can be answered locally using this set of materialized views.
3. The combination of the query evaluation cost and the view maintenance cost (*operational cost*) is minimal.

Based on the transformation rules, we model the DW design problem as a state space search problem. A solution to the problem can be found (if a solution to the problem exists) by defining equivalent states and exhaustively enumerating all equivalence classes.

Our approach is implemented for a class of SPJ-queries. We design algorithms that significantly prune the search space using cost-based and other heuristics, and we report on their experimental evaluation

This approach was initially adopted in [12] for a *restricted class* of selection-join queries and it is extended here by: (a) considering space constraints in the formulation of the DW design problem, (b) considering a broader class of queries involving projections and "pure" Cartesian products, and (c) implementing the approach and testing the new algorithms experimentally.

Outline. The rest of the paper is organized as follows. The next section contains related work. In Section 3, we formally state the DW design problem after providing some basic definitions. Section 4 introduces the transformation rules. In Section 5 the DW design problem is modeled as a state space search problem. Section 6 presents pruning algorithms and implementation issues. Experimental results are provided in Section 7. Finally, Section 8 contains concluding remarks and possible extension directions.

2 Related Work

The view selection problem has been addressed by many authors in different contexts. [5] provides algorithms for selecting a set of views to materialize that minimizes the query evaluation cost, under a space constraint, in the context of aggregations and multidimensional analysis. This work is extended in [4] where greedy algorithms are provided, in the same context, for selecting both views and indexes. In [3] greedy algorithms are provided for the view selection problem when queries are represented as AND/OR expression dags for multiple queries.

In [9], given a materialized SQL view, an exhaustive approach is presented as well as heuristics for selecting additional views that optimize the total view maintenance cost. [6] considers the same problem for select-join views and indexes. Space considerations are also discussed. Given an SPJ view, [8] derives, using key and referential integrity constraints, a set of auxiliary views other than the base relations that eliminate the need to access the base relations when maintaining both the initial and the auxiliary views (i.e. that makes the views altogether self-maintainable).

[10, 14] aim at optimizing the combined query evaluation and view maintenance cost: [10] provides an A* algorithm in the case where views are seen as sets of pointer arrays under a space constraint. [14] considers the same problem for materialized views but without space constraints. [3] provides a formalization of the problem of selecting a set of views that minimizes the combined cost under a space constraint but it does not provide any algorithm for solving the problem in the general case. [14, 3] consider a DW environment where all the source relations are available locally for computation.

None of the previous approaches requires the queries to be answerable exclusively from the materialized views. This requirement is taken into account in [12, 13] where the view selection problem is addressed with ([13]) and without ([12]) space constraints, but for a class of selection-join queries.

3 Formal statement of the DW design problem

We consider that a non-empty set of queries $\mathbf{Q}$ is given, defined over a set of source relations $\mathbf{R}$. The DW contains a set of materialized views $\mathbf{V}$ over $\mathbf{R}$ such that every query in $\mathbf{Q}$ can be rewritten completely over $\mathbf{V}$ [7]. Thus, all the queries in $\mathbf{Q}$ can be answered locally at the DW, without accessing the source relations in $\mathbf{R}$. Let Q be a query over $\mathbf{R}$. By Q^V, we denote a complete rewriting of Q over $\mathbf{V}$.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 $< \mathbf{V}, \mathbf{Q}^V >$.

Consider a DW configuration $\mathbf{C} =< \mathbf{V}, \mathbf{Q}^V >$. The *query evaluation cost* of the queries in $\mathbf{Q}^V$, denoted $E(\mathbf{Q}^V)$, is the weighted sum of the evaluation cost of each query rewriting in $\mathbf{Q}^V$. The view maintenance cost of the views in $\mathbf{V}$, denoted $M(\mathbf{V})$, is the weighted sum of the view maintenance cost of each view in $\mathbf{V}$ in the presence of the other views in $\mathbf{V}$. Note that the maintenance cost of a view depends on the presence of other views in the DW [9, 8]. The *operational cost* of $\mathbf{C}$, denoted $T(\mathbf{C})$, is $cE(\mathbf{Q}^V) + M(\mathbf{V})$. The parameter c indicates the relative importance of the query evaluation and view maintenance cost. The *storage space* needed for materializing the views in $\mathbf{V}$, denoted $S(\mathbf{V})$, is the sum of the space needed for materializing each view in $\mathbf{V}$.

The *DW design problem* can now be stated as follows.

Input: A set of queries $\mathbf{Q}$.
The functions: E for the query evaluation cost, M for the view maintenance cost, and S for the materialized views space.
The space available in the DW for materialization t.
A parameter c.

Output: A DW configuration $\mathbf{C} =< \mathbf{V}, \mathbf{Q}^V >$ such that $S(\mathbf{V}) \leq t$, and $T(\mathbf{C})$ is minimal.

4 Transformation rules

In order to model the DW design problem as a state space search problem we introduce a set of DW configuration transformation rules for a class of SPJ views. These rules operate on a graph representation for multiple views.

4.1 Multiquery graphs

We consider the class of relational expressions involving projection, selection and join operations that are equivalent to relational expressions of the standard form $\pi_X \sigma_F(R_1 \times \ldots \times R_k)$. The R_is $i \in [1, k]$, denote relations. The formula F is a conjunction of atoms (atomic formulas) of the form A *op* $B + c$ or A *op* B where *op* is one of the comparison operators $=$, $<$, $\leq$, $>$, $\geq$ (but no $\neq$), c is an integer valued constant and A, B are attributes from the R_i's. Atoms involving attributes from only one relation are called *selection atoms*, while those involving attributes from two relations are called *join atoms*. All the R_i's are distinct. X

is a non-empty set of attributes from the R_is. Queries and views considered here belong to this class.

Query rewritings over the views are in addition allowed to contain self-joins (and attribute renaming if the approach to the relational model that uses attribute names is followed).

A view can be represented by a *query graph*. Consider a view $V = \pi_X \sigma_F(R_1 \times \ldots \times R_k) \in \mathbf{V}$. The query graph G^V for V is a node and edge labeled multigraph defined as follows:

1. The set of nodes of G^V is the set of relations $R_1, \ldots, R_k$. Thus, in the following, we identify nodes with source relations. Every node R_i, $i \in [1, k]$, is labeled by an *attribute label for view* V. This is the (possibly empty) set of attributes of R_i in X, prefixed by V. An attribute label $V : *$ on a node R_i denotes that all the attributes of R_i are projected out in the view definition of V.
2. For every selection atom p in F involving one or two attributes of relation R_i, there is a loop on R_i in G^V labeled as $V : p$.
3. For every join atom p in F involving attributes of the relations R_i and R_j there is an edge between R_i and R_j in G^V labeled as $V : p$.

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 node and edge labeled multigraph resulting by merging the identical nodes of the query graphs for the views in $\mathbf{V}$. The label of a node R_i in $\mathbf{G}^V$ is the set containing the attribute labels of R_i in the query graphs for the views in $\mathbf{V}$. The edges of $\mathbf{G}^V$ are the edges in the query graphs for the views in $\mathbf{V}$.

Example 1. Consider the relations $R(K, A, B)$, $S(C, D)$, $T(E, F)$, and $U(G, H)$. Let V_1 and V_2 be two views over these relations defined as follows:
$V_1 = \pi_{KAE}(\sigma_{B<E+2}(\sigma_{K=A}(R) \bowtie_{B=C} \sigma_{C<D}(S) \bowtie_{D\leq E+2} \pi_E(T)))$, and
$V_2 = \pi_{EH}(\sigma_{C\leq D}(S) \bowtie_{D<E} T \bowtie_{E\geq G} \sigma_{H=3}(U))$.
Let $\mathbf{V} = \{V_1, V_2\}$. The multiquery graph $\mathbf{G}^V$ is depicted in Figure 1. This multiquery graph is used as a running example in this paper.

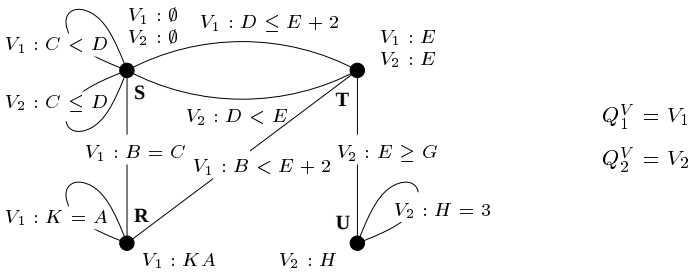


Fig. 1. The multiquery graph for $\mathbf{V}$, and a rewriting of $\mathbf{Q}$ over $\mathbf{V}$

Consider now the query set $\mathbf{Q} = \{Q_1, Q_2\}$, where
$Q_1 = \pi_{KAE}(\sigma_{K=A \wedge B=C \wedge C<D \wedge D \leq E+2 \wedge B<E+2}(R \times S \times T))$, and
$Q_2 = \pi_{EH}(\sigma_{C \leq D \wedge D<E \wedge E \geq G \wedge H=3}(S \times T \times U))$.
The queries in $\mathbf{Q}$ can be rewritten over $\mathbf{V}$ as shown in Figure 1. Thus Figure 1 shows the DW configuration $< \mathbf{G}^V, \mathbf{Q}^V >$, where $\mathbf{Q}^V = \{Q_1^V, Q_2^V\}$. This is the DW configuration $< \mathbf{G}^Q, \mathbf{Q}^Q >$.

4.2 Transformation rules

We now present five *transformation rules* that can be applied to a DW configuration $< \mathbf{G}^V, \mathbf{Q}^V >$. Each transformation rule consists of two parts. A part that transforms the multiquery graph $\mathbf{G}^V$ and a part that transforms the query set $\mathbf{Q}^V$ by rewriting queries in $\mathbf{Q}^V$ over the new view set. Note that $\mathbf{Q}^V$ is not necessarily the rewriting of $\mathbf{Q}$ over $\mathbf{V}$ that yields the optimal query evaluation cost.

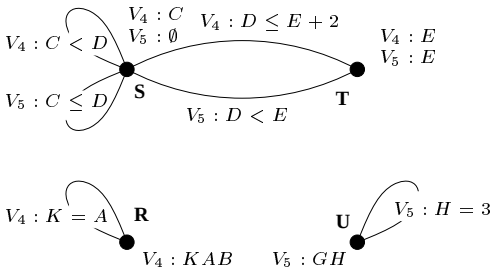
Transformation Rule 1 (Edge Removal). Let V be a view in $\mathbf{G}^V$, and e be an edge between nodes R_i and R_j labeled as $V : p$. Nodes R_i and R_j may coincide in which case e is a loop. Then,

1. $\mathbf{G}^V$ *transformation:*
   (a) If p is not implied by the rest of the atoms of V, replace in $\mathbf{G}^V$ the attribute label $V : Y_i$ ($V : Y_j$) of R_i (R_j) by $V : Y_i \cup \{A\}$ ($V : Y_j \cup \{A\}$), where A is the attribute of R_i (R_j) occuring in p. In the particular case where R_i and R_j coincide in R_i, replace in $\mathbf{G}^V$ the attribute label $V : Y_i$ of R_i by $V : Y_i \cup Y$, where Y is the set of the attribute or the attributes of R_i occuring in p.
   (b) Replace every occurrence of V in $\mathbf{G}^V$ by a new view name V_1. New view names should not already appear in $\mathbf{G}^V$.
   (c) Remove e from $\mathbf{G}^V$.
2. $\mathbf{Q}^V$ *transformation:*
   Let X be the set of all the attributes appearing in attribute labels for V in $\mathbf{G}^V$. If p is not implied by the rest of the atoms of V, replace any occurrence of V in $\mathbf{Q}^V$, by the expression $\pi_X(\sigma_p(V_1))$. Otherwise, replace any occurrence of V in $\mathbf{Q}^V$, by V_1.

Example 2. Figure 2 illustrates the DW configuration resulting by three applications of rule 1, in sequence, to the edges labeled as $V_1 : B < E+2$, $V_1 : B = C$, and $V_2 : E \geq G$ of the multiquery graph depicted in Figure 1.

Transformation Rule 2 (Attribute Removal). Let V be a view in $\mathbf{G}^V$, X be the set of all the attributes appearing in attribute labels for V, and $A_1, \ldots, A_k, B_1, \ldots, B_k$ be attributes in X. If the atoms $A_1 = B_1, \ldots, A_k = B_k$, where $A_i \neq A_j$, and $A_i \neq B_j$, for every $i, j \in [1, k]$, are implied by the set of the atoms of V, then,

1. $\mathbf{G}^V$ *transformation:*
   (a) Replace every attribute label $V : Y_i$ in $\mathbf{G}^V$ by the attribute label $V : (Y_i - \{A_1, \ldots, A_k\})$.
   (b) Replace every occurrence of V in $\mathbf{G}^V$ by a new view name V_1.

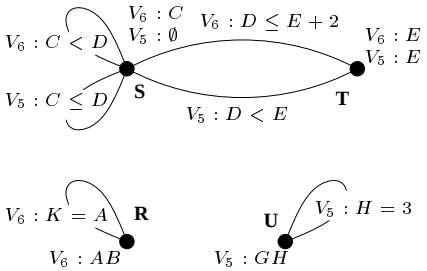


$$Q_1^V = \pi_{KAE}(\sigma_{B=C}(V_4))$$
$$Q_2^V = \pi_{EH}(\sigma_{E \geq G}(V_5))$$

Fig. 2. A DW configuration resulting by three applications of rule 1 (edge removal)

2. $\mathbf{Q}^V$ *transformation:*
   Replace any occurrence of V in $\mathbf{Q}^V$, by $V_1 \bowtie_{A_1=B_1} \delta_{B_1 \to A_1}(\pi_{B_1}(V_1)) \bowtie_{A_2=B_2} \ldots \bowtie_{A_k=B_k} \delta_{B_k \to A_k}(\pi_{B_k}(V_1))$. The operator δ denotes the attribute renaming operator.

Example 3. The atom $K = A$ of view V_4 is implied by the set of the atoms of V_4 (it is even present in the set), and K and A occur in attribute labels for V_4. Thus rule 2 can be applied to view V_4 of Figure 2. The result of this application is depicted in Figure 3.



$$Q_1^V = \pi_{KAE}(\sigma_{B=C}(V_6 \bowtie_{A=K} \delta_{A \to K}(\pi_A(V_6))))$$
$$Q_2^V = \pi_{EH}(\sigma_{E \geq G}(V_5))$$

Fig. 3. A DW configuration resulting by an application of rule 2 (attribute removal)

Transformation Rule 3 (View Break). Let V be a view and N_1 and N_2 be two sets of nodes labeled by V, in $\mathbf{G}^V$, such that:

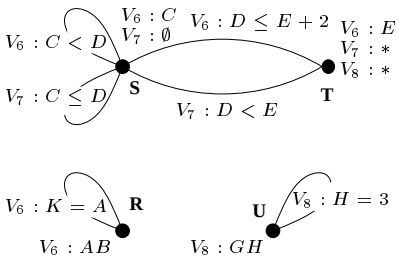
(a) $N_1 \not\subseteq N_2$ and $N_2 \not\subseteq N_1$.
(b) $N_1 \cup N_2$ is the set of all the nodes labeled by V in $\mathbf{G}^V$.
(c) There is no edge labeled by V in $\mathbf{G}^V$, between a node in $N_1 - N_2$ and a node in $N_2 - N_1$.

1. $\mathbf{G}^V$ *transformation:*
   (a) Let V_1 and V_2 be new view names. To the label of every node in $N_1 - N_2$ ($N_2 - N_1$), containing the attribute label $V : Y$, in $\mathbf{G}^V$, add the attribute label $V_1 : Y$ ($V_2 : Y$). To the label of every node in $N_1 \cap N_2$, in $\mathbf{G}^V$, add the attribute labels $V_1 : *$ and $V_2 : *$.

(b) For every edge between nodes in N_1 (N_2) in $\mathbf{G}^V$, labeled as $V : p$, add an edge between the same nodes labeled as $V_1 : p$ ($V_2 : p$).
(c) Remove from $\mathbf{G}^V$ all the attribute labels for V, and all the edges labeled by V.

2. $\mathbf{Q}^V$ *transformation:*
Let X be the set of all the attributes appearing in attribute labels for V. Replace any occurrence of V in $\mathbf{Q}^V$, by $\pi_X(V_1 \bowtie V_2)$. The operator $\bowtie$ (without subscript) denotes the natural join operator.

Example 4. Consider the DW configuration of Figure 3. Rule 3 can be applied to view V_5 for $N_1 = \{S, T\}$ and $N_2 = \{T, U\}$. The resulting DW configuration is depicted in Figure 4.



$$Q_1^V = \pi_{KAE}(\sigma_{B=C}(V_6 \bowtie_{A=K} \delta_{A\to K}(\pi_A(V_6))))$$
$$Q_2^V = \pi_{EH}(\sigma_{E\geq G}(V_7 \bowtie V_8))$$

Fig. 4. A DW configuration resulting by an application of rule 3 (view break)

Transformation Rule 4 (View Merging). Let V_1 and V_2 be two views in $\mathbf{G}^V$, and N_1 and N_2 be their sets of nodes respectively such that:
(a) There is no edge labeled by V_1 (V_2) between a node in $N_1 - N_2$ ($N_2 - N_1$) and a node in $N_1 \cap N_2$.
(b) For every atom p of V_1 (V_2) labeling an edge between nodes in N_1 (N_2), p implies or is implied by an atom of V_2 (V_1).

Then, let $\mathbf{F}$ be the set of the atoms p of V_1 such that p is implied by an atom of V_2 and is not implied by a different atom of V_1 and of the atoms p of V_2 such that p is implied by an atom of V_1 and is not implied by a different atom of V_2. Let $\mathbf{F}_1$ ($\mathbf{F}_2$) be the set of all the atoms of V_1 (V_2) in $\mathbf{G}^V$ that are not implied by $\mathbf{F}$, and X be the set of attributes occurring in $\mathbf{F}_1$ or $\mathbf{F}_2$.

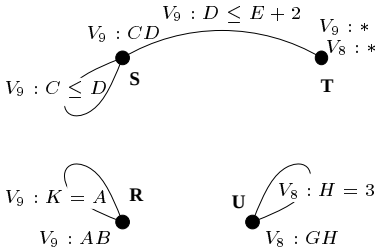
1. $\mathbf{G}^V$ *transformation:*
(a) Let V be a new view name. To the label of every node R in $N_1 \cap N_2$ containing the attribute labels $V_1 : Y_1$ and $V_2 : Y_2$, add an attribute label $V : (Y \cup Y_1 \cup Y_2)$, where Y is the set of all the attributes of R in X.
To the label of every node in $N_1 - N_2$ ($N_2 - N_1$) containing the attribute label $V_1 : Y_1$ ($V_2 : Y_2$), add the attribute label $V : Y_1$ ($V : Y_2$).
(b) For every atom p in $\mathbf{F}$, labeling an edge between nodes R_i and R_j (R_i and R_j may coincide), add to $\mathbf{G}^V$ an edge between R_i and R_j labeled as $V : p$.

For every atom p of V_1 (V_2) labeling an edge between nodes in $N_1 - N_2$ ($N_2 - N_1$), add to $\mathbf{G}^V$ an edge between the same nodes labeled as $V : p$.

(c) Remove from $\mathbf{G}^V$ all the attribute labels for V_1 or V_2 and all the edges labeled by V_1 or V_2.

2. $\mathbf{Q}^V$ *transformation:*
Replace any occurrence of V_1 (V_2) in $\mathbf{Q}^V$ by $\pi_{X_1}(\sigma_{F_1}(V))$ ($\pi_{X_2}(\sigma_{F_2}(V))$), where X_1 (X_2) is the set of all the attributes in attribute labels for V_1 (V_2), and F_1 (F_2) is the conjunction of the atoms in $\mathbf{F}_1$ ($\mathbf{F}_2$).

Example 5. Consider the DW configuration of Figure 4. Rule 4 can be applied to views V_6 and V_7 for $N_1 = \{R, S, T\}$ and $N_2 = \{S, T\}$. N_1 and N_2 satisfy the premise of the rule. Its application to the DW configuration of Figure 4 results in the DW configuration depicted in Figure 5.



$$Q_1^V = \pi_{KAE}(\sigma_{B=C}(\pi_{ABCE}(\sigma_{C<D}(V_9)) \bowtie_{A=K} \delta_{A\to K}(\pi_A(\pi_{ABCE}(\sigma_{C<D}(V_9))))))$$

$$Q_2^V = \pi_{EH}(\sigma_{E\geq G}(\pi_{EF}(\sigma_{D<E}(V_9)) \bowtie V_8))$$

Fig. 5. A DW configuration resulting by an application of rule 4 (view merging)

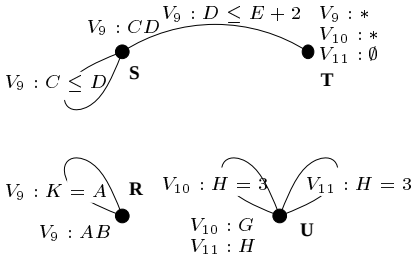
Transformation Rule 5 (Attribute Transfer). Let V be a view in $\mathbf{G}^V$, X be the set of all the attributes appearing in attribute labels for V, and $\{A_1, \ldots, A_k\}$ be a proper subset of X. If the atoms $A_1 = c_1, \ldots, A_k = c_k$, where c_i is a constant, $i = 1, \ldots, n$, are implied by the set of atoms of V, then,

1. $\mathbf{G}^V$ *transformation:*
(a) Let V_1 and V_2 be new view names. To the label of every node containing the attribute label $V : Y$ in $\mathbf{G}^V$ add the attribute labels $V_1 : (Y - \{A_1, \ldots, A_k\})$ and $V_2 : (Y \cap \{A_1, \ldots, A_k\})$.
(b) For every edge labeled as $V : p$ in $\mathbf{G}^V$ add two edges between the same nodes labeled as $V_1 : p$ and $V_2 : p$.
(c) Remove from $\mathbf{G}^V$ all the attribute labels for V, and all the edges labeled by V.

2. $\mathbf{Q}^V$ *transformation:*
Replace any occurrence of V in $\mathbf{Q}^V$, by $V_1 \times V_2$.

Example 6. Consider the DW configuration of Figure 5. The atom $H = 3$ is implied by the set of atoms of V_8 in $\mathbf{G}^V$ (it is even present in this set). The DW configuration in Figure 6 results by the application of rule 5 to V_8, for $k = 1$, and $A_1 = H$.

The set of the previous transformation rules is sound, complete and minimal [11].



$$Q_1^V = \pi_{KAE}(\sigma_{B=C}(\pi_{ABCE}(\sigma_{C<D}(V_9)) \bowtie_{A=K} \delta_{A\to K}(\pi_A(\pi_{ABCE}(\sigma_{C<D}(V_9))))))$$

$$Q_2^V = \pi_{EH}(\sigma_{E\geq G}(\pi_{EF}(\sigma_{D<E}(V_9)) \bowtie (V_{10} \times V_{11})))$$

Fig. 6. A DW configuration resulting by an application of rule 5 (attribute transfer)

5 The DW design problem as a state space search problem

We now model the DW design problem as a state space search problem. We define states and transitions between states.

A *state* s is a DW configuration $\mathbf{C} =< \mathbf{G}^V, \mathbf{Q}^V >$. A particular state, the state $< \mathbf{G}^Q, \mathbf{Q}^Q >$ is called initial state and is denoted s_0.

With every state s a cost is associated through the function $cost(s)$. This is the operational cost $T(\mathbf{C})$ of the DW configuration $\mathbf{C}$. Also, a size is associate through the function $size(s)$. This is the space $S(\mathbf{V})$ needed for materializing the views in $\mathbf{V}$.

There is a transition $T(s, s')$ from state $s =< \mathbf{G}^V, \mathbf{Q}^V >$ to state $s' = < \mathbf{G}^{V'}, \mathbf{Q}^{V'} >$ if $< \mathbf{G}^{V'}, \mathbf{Q}^{V'} >$ can be obtained by applying any of the five transformation rules to $< \mathbf{G}^V, \mathbf{Q}^V >$.

Two states s and s' are *equivalent* if there is a renaming of the views in $\mathbf{V}$ that makes $\mathbf{G}^V$ and $\mathbf{G}^{V'}$ identical, and Q_i^V and $Q_i^{V'}$ equivalent, for every $Q_i^V \in \mathbf{Q}^V$.

Viewing the states as nodes and the transitions between them as directed edges, the *search space* is a directed graph determined by the initial state and the states we can reach from it following transitions in all possible ways. Equivalent states are represented in the search space by the same node. Clearly the search space is a rooted at s_0 directed acyclic graph.

The cost and the size of a state is computed *incrementally* when transiting from one state to another. Usually, a transformation rule affects a small number of views in $\mathbf{G}^V$ and of query rewritings in $\mathbf{Q}^V$. Thus, the incremental computation is a substantial improvement in the computation of the cost and a size of new states.

6 Implementation and algorithms

Our approach is implemented for a class of SPJ-queries. For simplicity, queries and views are as defined in Section 4 but do not contain 'pure' Cartesian products. Query rewritings over the materilalized views are of the same form too. The maintenance cost model takes into account the cost of transmitting data between the DW and the data sources, the cost of computing view changes and

the cost of applying the changes to the materialized views. We consider that each view is maintained separately.

An exhaustive algorithm can be very expensive for a large number of complex queries. We outline now algorithms that significantly prune the search space.

Algorithm 1. Pruning algorithm

1. Consider the states $s_1 =< \mathbf{G}^{Q_1}, \mathbf{Q}_1^{Q_1} >, \ldots, s_n =< \mathbf{G}^{Q_n}, \mathbf{Q}_n^{Q_n} >$, where $\mathbf{Q}_1 = \{Q_1\}, \ldots, \mathbf{Q}_n = \{Q_n\}$. For every s_i, and every subset E of edges in $\mathbf{G}^{Q_i}$, we generate new states by applying rule 1 (edge removal) to all the edges in E. From the states thus created, we generate new states by applying exhaustively rule 3 (view break).
   $S_i = \{s_i^1, \ldots, s_i^{k_i}\}$ denotes the set of states created from s_i, $i = 1, \ldots, n$.
2. Consider two states. By combining these states we can create a new state. The combined state is formed by merging their multiquery graphs and unionning their query rewriting sets.
   The second part of the algorithm is explained through the use of a tree depicted in Figure 7. This tree is defined as follows: the nodes of the tree are states and combined states, and the children of the root node are the states $s_1^1, \ldots, s_1^{k_1}$. These nodes are at depth 1. The children of a node s which are at depth d, $d > 1$, are the combined states resulting by combining s with eah of the states $s_d^1, \ldots, s_d^{k_d}$ and all the states resulting by applying transformation rule 4 (view merging) to these combined states (not necessarily once). The leaves of the tree (nodes at depth n) are states (DW configurations) $< \mathbf{G}^Q, \mathbf{Q}^V >$. The algorithm proceeds with the generation of the tree in a depth-first manner. It keeps the state $s_m =< \mathbf{G}^Q, \mathbf{Q}^V >$ that satisfies the space constraint and has the lowest cost, among those generated, along with its cost c_m. s_m and c_m is initially set to $\emptyset$ and ∞ respectively. *The generation of the tree is discontinued below a node if this node does not satisfy the space constraint or if its cost exceeds* c_m. At the end of the process the algorithm returns s_m.

Even though a pruning of the search space is performed, Algorithm 1 always computes a state that satisfies the space constraint and has minimal cost (if it exists). This is due to the fact that, under the assumptions we have made, the cost and the size of a node in the tree depicted in figure 7 are not less than the cost and the size of his descendant nodes.

Algorithm 2. Greedy algorithm

1. The first part is similar to the first part of the pruning algorithm.
2. In this part the algorithm starts by considering the root node of the tree of Figure 7. When a node is considered, all its child-nodes are generated and one with the minimal cost among those that satisfy the space constraint is chosen for consideration. The algorithm stops when no child-node satisfies the space constraint (and returns no answer) or when a leaf node s_m is chosen for consideration (and returns s_m).

Clearly this algorithm is efficient. It may though return no answer even if a solution to the problem exists.

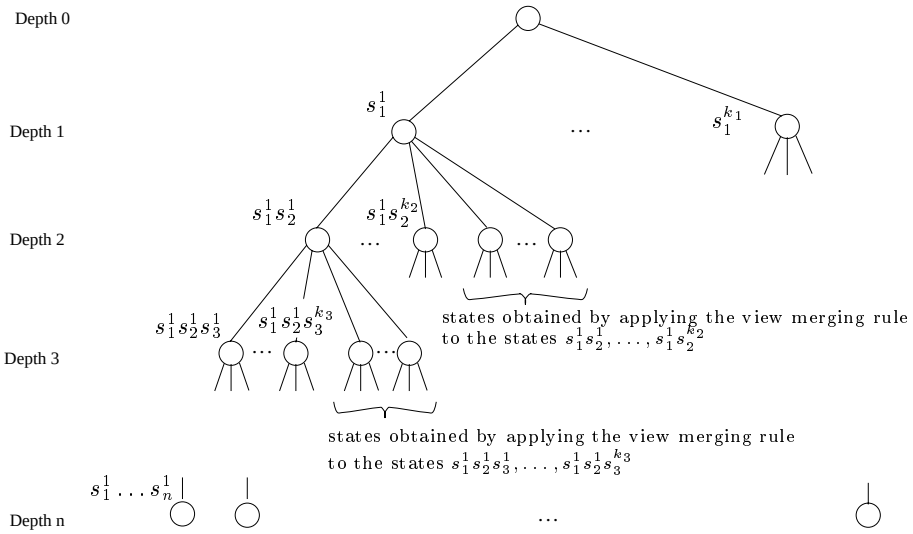


Fig. 7.

Algorithm 3. Heuristic algorithm

1. The first part is similar to the first part of the pruning algorithm.
2. We keep in each $S_i,\ i \in [1,n]$, only those states s_i that satisfy the space constraint and have minimal cost and those states that when combined with another state $s_j \in S_j,\ i \leq j$, allow the application of the view merging rule to the combined state.
3. This part is similar to the second part of the pruning algorithm.

This algorithm always returns the optimal solution if there are no limitations to the space allocated to the DW. The reason is that the optimal solution is obtained either by combining one optimal state from each $S_i,\ i \in [1,n]$ or by combining one state from each $S_i,\ i \in [1,n]$ and applying at least once the view merging rule to an intermediate combined state.

7 Experimental results

We have performed a sequence of experiments to compare the algorithms presented in the previous section. The comparison is first made with respect to the time required to find the solution. The time required by each algorithm is expressed as CPU time. We also study the greedy algorithm with respect to the quality of the solution returned. This is expressed as the percentage of the cost of the optimal state returned by the pruning algorithm divided by the cost of the final state that the greedy algorithm returns.

The algorithms are compared in terms of the following factors: (a) the complexity of the input query set, and (b) physical factors. The complexity of the query set is expressed by three parameters: the number of input queries, the number of selection and join edges of all the input queries, and the overlapping of the queries in the input query set. The query overlapping is expressed by

the total number of implications between the selection or join atoms of different input queries. The physical factors are expressed by the relative importance of the query evaluation and view maintenance cost.

In the following we describe in detail the various experiments performed.

Experiment 1 : The number of queries varies. We study the performance of the exhaustive algorithm when the number of the input queries varies. We consider that there are no space constraints, and that the number of edges per query is constant. The percentage of implications (number of implications per number of edges) does not significantly vary (Figures 8.a and b).

Experiment 2 : The number of edges varies. We study the performance of the algorithms when the total number of selection and join edges of the input queries varies. The experiment was performed with 4 queries. We consider that there are no space constraints, and that percentage of implications does not significantly vary. (Figures 8.c and d).

Experiment 3 : The number of implications varies. We ran the algorithms for a set of 3 queries, each one containing 4 atoms while varying the number of implication between atoms of different queries. The space available is unlimited. (Figures 8.e and f).

Experiment 4 : Parameter c varies. We ran the algorithms for a set of 4 queries, each one containing 4 atoms, while varying the parameter c. The percentage of implications is kept constant. (Figures 8.g and h).

From the previous experiments, we conclude that the heuristic algorithm is the winner when we have no significant restrictions in the space available for materialization. The pruning algorithm is our proposal in cases where significant space restrictions are imposed. The greedy algorithm is a very fast alternative, but it suffers from the fact that in many cases it does not find solutions which are produced by the application of the view merging transformation. Thus, it returns acceptable solutions in the cases where we have limited overlapping between queries, or where the view maintenance cost or the query evaluation cost are important.

8 Conclusion and possible extensions

In this paper we have dealt with the issue of selecting a set of views to materialize in a DW. This decision is important when implementing a DW. Current products do not provide tools for automatic DW design.

We provide a generic method that, given a set of SPJ queries to be answered by the DW, generates all the "significant" materialized view selections that answer all the input queries. This is done through the use of a set of transformation rules which detect and exploit common subexpressions between the queries. The rules modify view selections and rewrite completely all the input queries over the modified view selections. Then we address the problem of selecting a set of views that fits in the space available at the DW for materialization, and minimizes the combined view maintenance and query evaluation cost. Using the transformation rules we model the previous problem as a state space search problem and

we design and implement various algorithms that heuristically prune the search space. These algorithms are compared through experimental evaluation.

Our future work includes extending the approach to handle a larger class of queries and views, and in particular grouping/aggregation queries. This is an important issue in the design of Data Marts where grouping/aggregation queries are extensively used for On-Line Analytical Processing and Decision Support.

We are currently working on DW evolution issues. DWs are entities that evolve over time. Dynamic interpretations of the DW design problem, where the DW is incrementally designed when initial parameters to the problem are modified (for instance the input query set), is an important issue and it is at the focus of our research efforts.

References

[1] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1):65–74, 1997.

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

[3] 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.

[4] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index Selection for OLAP. In *Proc. of the 13th Intl. Conf. on Data Engineering*, pages 208–219, 1997.

[5] 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.

[6] W. Labio, D. Quass, and B. Adelberg. Physical Database Design for Data Warehousing. In *Proc. of the 13th Intl. Conf. on Data Engineering*, 1997.

[7] 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.

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

[9] 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.

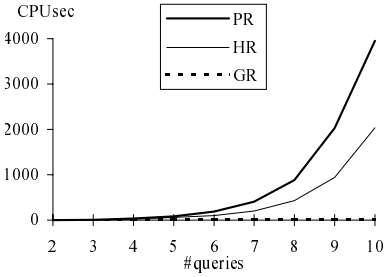
[10] N. Roussopoulos. View Indexing in Relational Databases. *ACM Transactions on Database Systems*, 7(2):258–290, 1982.

[11] D. Theodoratos, S. Ligoudistianos, and T. Sellis. Designing the Global DW with SPJ Queries. *Technical Report, Knowledge and data Base Systems Laboratory, Electrical and Computer Engineering Dept., National Technical University of Athens*, Nov. 1998.

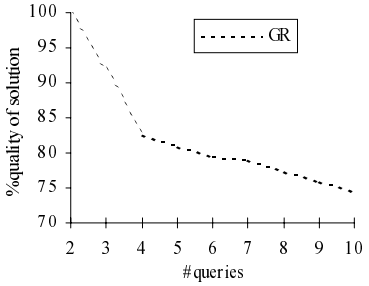
[12] D. Theodoratos and T. Sellis. Data Warehouse Configuration. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 126–135, 1997.

[13] D. Theodoratos and T. Sellis. Data Warehouse Schema and Instance Design. In *Proc. of the 17th Intl. Conf. on Conceptual Modeling (ER'98)*, 1998.

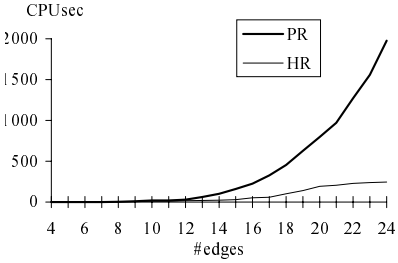
[14] J. Yang, K. Karlapalem, and Q. Li. Algorithms for Materialized View Design in Data Warehousing Environment. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 136–145, 1997.



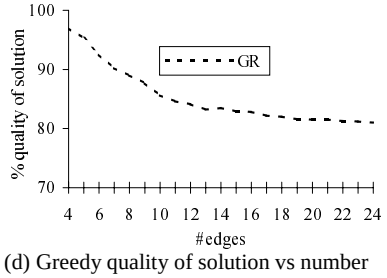
(a) CPU time vs size of input



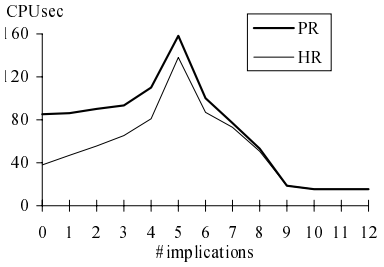
(b) Greedy quality of solution vs size of input



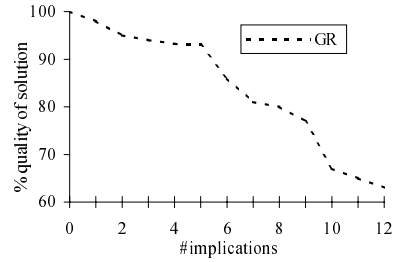
(c) CPU time vs number of edges



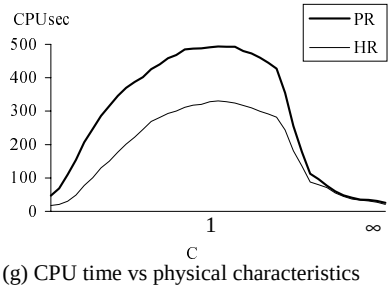
(d) Greedy quality of solution vs number of edges



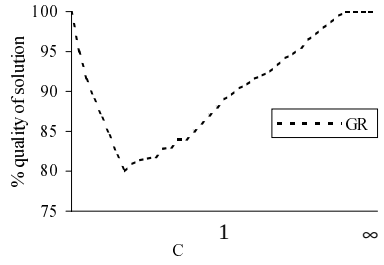
(e) CPU time vs number of implications



(f) Greedy quality of solution vs number of implications



(g) CPU time vs physical characteristics



(h) Greedy quality of solution vs physical characteristics

Fig. 8. Experimental results