A Randomized Approach for the Incremental Design of an Evolving Data Warehouse

Dimitri Theodoratos, Theodore Dalamagas, Alkis Simitsis, and Manos Stavropoulos

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

Abstract. A Data Warehouse (DW) can be used to integrate data from multiple distributed data sources. A DW can be seen as a set of materialized views that determine its schema and its content in terms of the schema and the content of the data sources. DW applications require high query performance. For this reason, the design of a typical DW consists of selecting views to materialize that are able to answer a set of input user queries. However, the cost of answering the queries has to be balanced against the cost of maintaining the materialized views. In an evolving DW application, new queries need to be answered by the DW. An incremental selection of materialized views uses the materialized views already in the DW to answer parts of the new queries, and avoids the re-implementation of the DW from scratch. This incremental design is complex and an exhaustive approach is not feasible. We have developed a randomized approach for incrementally selecting a set of views that are able to answer a set of input user queries locally while minimizing a combination of the query evaluation and view maintenance cost. In this process we exploit "common sub-expressions" among new queries and between new queries and old views. Our approach is implemented and we report on its experimental evaluation.

1 Introduction

A Data Warehouse (DW) is a repository of information integrated from multiple data sources. A DW is devoted to On-Line Analytical Processing (OLAP) and Decision Support System (DSS) applications [30,15,11,2]. Current DWs accumulate enormous quantities of data. OLAP queries are complex data intensive queries involving grouping/aggregation and sequential scans. Therefore, query evaluation in the DW has to be efficient in order to guarantee fast answers to the analysts. The data stored in a DW need to be eventually updated in response to updates of the source data.

A DW is usually seen as a set of materialized views defined over the relations of the data sources. These views determine its schema and content in terms of the schema and the content of the data sources. In order to achieve high query performance, the queries are evaluated using exclusively the materialized views. The query evaluation cost of a set of queries is the cost of evaluating these queries rewritten completely [18,7,4,3] over the materialized views. The materialized views are usually maintained incrementally. In an incremental strategy, only the changes that must be applied to the views are computed from the changes of the source relations [8,6,19]. The DW view maintenance cost is the cost of propagating the source relation changes to the materialized views that are affected by these changes.

A central problem in the design of a DW is the selection of views to materialize given a set of input queries that the DW has to satisfy. When designing a DW, one of the most significant challenges is the minimization of the query evaluation cost. However, maintaining the materialized views is a particularly time consuming process that makes part of the data unavailable for querying. Therefore, in order to ensure sufficient availability of the stored data and satisfactory query response time, a reduction in the query evaluation cost must be balanced against an increase in the view maintenance cost. Many approaches attempt to appropriately select views for materialization that exploit common sub-expressions between the input queries and minimize a combination of the query evaluation and the view maintenance cost [9,27,1,31,28,26]. This last solution, though more complex, achieves a balance between query performance and availability of the data.

The problem. DWs are dynamic and evolve over time. As the needs of the analysts expand, new queries need to be satisfied by the DW. Some of these new queries can be answered by the views that are already materialized in the DW. In general though, new materialized views need to be added to the DW either for answering the new queries or for improving the evaluation performance of the new queries. The new queries are then answered through rewritings over the old views, or over the new views, or partially over the new and partially over the old views. Re-implementing the whole DW from scratch for dealing with the new queries is complex and implies the unavailability of the data for a long period of time. In [29] we avoid re-implementation and present an incremental approach that selects new views for materialization that minimizes the combined cost of evaluating the new queries and maintaining the new materialized views. We also provide transformation rules that generate alterative view selections for materialization and query evaluation plans for the queries from the views in the form of AND/OR dags. Each view in a view selection is used for evaluating at least one new query. The transformation rules are shown to be sound and complete. In this sense, by exhaustively applying the rules, the view selection that minimizes the combined cost can be found. Nevertheless, for a large number of complex queries, the search space can be very big. In real cases, it is unfeasible to use an exhaustive algorithm for exploring it.

Contribution. In this paper, we present a randomized approach for the incremental design of a DW that is, for the selection of new views to materialize in the DW. The approach is based on the Simulated Annealing process. Since its first appearance [16], Simulated Annealing has been used in a variety of optimization problems. In the Database area, it has been used for query optimization [14,24,23,13]. Even though it has been studied in research works, it has not been adopted in commercial products. The reason is that randomized algorithms such as Simulated Annealing, although they return far better results than heuristic or greedy algorithms, they require a longer running time [22]. In the incremental design of a DW, the time restrictions are not so crucial. Further, Simulated Annealing is especially well suited to optimization problems with large search spaces and with cost functions that manifest a large number of local minima [14] as is the case here. Thus, Simulated Annealing is a good candidate for incrementally designing a DW.

Some parameters of the simulated annealing algorithm depend on the problem addressed while others are implementation-dependent. For the problem in hand, we use multiquery AND/OR dags (AO dags) and transformations rules to define a search space suitable for applying the Simulated Annealing. The search space is a graph of states. The states of the search space (nodes of the graph) are multiquery AO dags. A move from one state to another in the search space (an edge of the graph) roughly corresponds to application of a transformation rule. The cost of each state is the combined cost of evaluating the new queries according to the evaluation plans represented in a state and maintaining new materialized views. Note that our approach does not depend on the way these costs are assessed.

We have implemented the Simulated Annealing approach for incrementally designing a DW. We have first run a number of experiments to choose the implementation dependent parameters. The approach is then applied for selecting views defined over a real-world star schema [15] comprising one fact table and three dimension tables with three to four hierarchy levels. The number of dimensions and hierarchy levels of this schema and the number of types of views are intentionally kept small so that the solution returned by our approach can be compared with that of the exhaustive algorithm when a number of the parameters of the problem vary.

Outline. The rest of the paper is organized as follows. The next section reviews related work. Section 3 introduces multiquery AO dags, states formally the incremental design problem in terms of multiquery AO dags, and describes the transformation rules. In Section 4 we briefly present the Simulated Annealing algorithm, and we show how the problem-dependent and implementation dependent parameters are defined for the incremental design problem. Section 5 deals with implementation issues and presents experimental results. We conclude in Section 6 and provide directions for further work.

2 Related Work

Although many works deal with the design of a DW, only few of them address dynamic issues. The works dealing with the design of a DW can be classified according to the class of queries and views considered, the cost function they attempt to minimize, the constraints that impose to the problem, and the approach adopted for solving it. [12] aims at minimizing the query evaluation cost in the context of aggregations and multidimensional analysis under a space constraint. In the same context, [1] proposes techniques that consider only the relevant elements in the search space of the problem.Given a materialized SQL view, [21] presents an exhaustive approach as well as heuristics for selecting auxiliary views that minimize the total view maintenance cost. Given an SPJ view, [20] 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). In [9] greedy algorithms are provided for selecting views to materialize that minimize the query evaluation cost under a space constraint. A integer programming solution for selecting views that minimize the combined cost is given in [31]. A variation of the DW design problem endeavoring to select a set of views that minimizes the query evaluation cost under a total maintenance cost constraint is adopted in [10]. None of the previous approaches requires the queries to be answerable exclusively from the materialized views in a non-trivial manner (that is, without considering that the source relations and the materialized views reside in the same site, and without replicating all the source relation at the DW). This requirement is taken into account in [27] where the problem of configuring a DW without space restrictions is addressed for a class of select-join queries. This work is extended in [28] in order to take into account space restrictions, multiquery optimization over the maintenance expressions, and the use of auxiliary views when maintaining other views. [26] deals with the same problem for a class of PSJ queries under space restrictions.

The approaches adopted by the previous papers are tailored for the static DW design problem. An incremental version of the DW design problem (dynamic DW design) is addressed in [29]. [17] presents a system that dynamically materializes information at multiple levels of granularity. However, this approach is not based on a given set of queries for selecting view materializations but it cashes the results of incoming queries in order to match the workload and maintenance restrictions. [25] deals with a problem that is somehow complementary to the incremental design problem: detect materialized views that become redundant in an evolving DW. We are not aware of any work using the Simulated Annealing process for the design of a DW. Papers using Simulated Annealing for optimizing queries are cited in the introduction.

3 Problem Formulation and Transformation Rules

We present in this section an AO dag representation for multiple queries and views. We then formulate the problem in terms of AO dags and describe informally the AO dag transformation rules introduced in [29].

3.1 Problem Formulation

Multiquery AO dags [5,21] provide a convenient, compact, representation of alternative ways of evaluating multiple queries from source relations and views and of subexpression sharing between relations and views. This representation is quite general to allow for complex queries including grouping/aggregation queries that are extensively used in DWing applications. It distinguishes between AND nodes that correspond to views (henceforth *view nodes*) and OR nodes that correspond to operations (henceforth *operation nodes*). An extension of these dags with *marked view nodes* [29] allows the representation of views already materialized in the DW (old views).

Consider the following source relations that form a star schema:

`Location(`<u>`Store`</u>`, City, Country, Continent)` abbreviated as `L`
`Product(`<u>`Item`</u>`, Type, Category)` abbreviated as `P`
`Time(`<u>`Date`</u>`, Month, Quarter, Year)` abbreviated as `T`
`Sales(`<u>`Store`</u>`, `<u>`Item`</u>`, `<u>`Date`</u>`, Amount)` abbreviated as `S`

`S` is a fact table comprising one measure attribute `Amount` and foreign key attributes to the dimension tables `L`, `P`, and `T`. The dimension tables comprise different hierarchy attributes. We have used this schema for applying our approach in the experiments.

As an example, a small multiquery AO dag representing alternative ways of evaluating three queries Q_1, Q_2, and Q_3 from the source relations is shown in Figure 1(a). These queries join the fact and the dimension tables and aggregate the resulting table at different aggregation levels. For simplicity only the fact table `S` and the dimension tables `L` and `P` are involved. In Figure 1(a), small circles

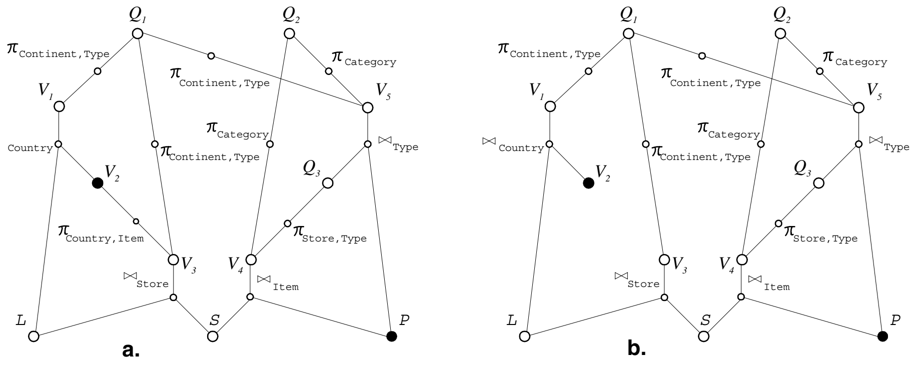

Fig. 1. (a) A multiquery AO dag for the queries Q_1, Q_2, and Q_3, (b) Initial multiquery AO dag.

represent operation nodes while bigger ones represent view nodes. An operation node is labeled by the corresponding operation, and a view node by a view name. Some view nodes correspond to the queries Q_1, Q_2 and Q_3. These view nodes are labeled by the names of the queries and are called *query nodes*. Marked view nodes (old materialized views) are shown by filled black circles (view nodes V_2 and `P` in Figure 1(a)). Only two types of operations are involved: (a) a binary one, the natural join operation, $\bowtie_A$, where A denotes the attribute on which the join is performed, and (b) a unary one, the generalized projection operation [7], $\pi_{X,\texttt{sum}(A)}$, where X is a set of grouping attributes from the dimension relations and A is a measure attribute aggregated by the aggregate function `sum`. For concreteness the aggregated measure $\texttt{sum}(A)$ is omitted in the figures. Three different alternative plans for query Q_1, two for query Q_2, and one for query Q_3 are depicted in the multiquery AO dag of Figure 1(a). Note that some views (expressions) are shared between the plans of different queries. For instance, view V_5 is shared between a plan of query Q_1 and a plan of query Q_2, while query Q_3 is a view node in an evaluation plan of query Q_2.

Some other AO dags for the queries Q_1, Q_2 and Q_3 are shown in Figures 1(b), 2(a), and 2(b). These multiquery AO dags are subdags of the multiquery AO dag of Figure 1(a). In contrast to the multiquery AO dag of Figure 1(a), in the multiquery AO dags of Figures 1(b), 2(a), and 2(b) the sink nodes are not necessarily source relations. In a multiquery AO dag for a set of queries, all the query nodes are present, and the same holds for the marked nodes. All the root nodes are query nodes (but a query node is not necessarily a root node). The sink nodes are view nodes and *represent materialized views.* Those of the sink nodes that are not marked are the *new views to materialize in the DW.* For instance in the multiquery AO dag of Figure 2(b), the materialized views are the views (or relations) `L`, `S`, `P`, V_2, and V_5, of which `L`, `S`, and V_5 are new views to materialize in the DW, and `P`, V_2 are views already materialized in the DW (old views). For each query node, *at least one evaluation plan from the sink nodes is represented.* Note that since the sink nodes represent (old and new) materialized views, *every query can be evaluated using the materialized views.* Every sink node that is not a marked node occurs in at least one query evaluation plan.

The incremental design problem that we address in this paper can now be formulated as follows. Consider a multiquery AO dag $\mathbf{G}$, for a set $\mathbf{Q}$ of new queries and a set $\mathbf{V}_0$ of old views, which represents alternative evaluation plans for the queries in $\mathbf{Q}$ over the source relations. $\mathbf{G}$ is given as input to the problem. The goal is to find a multiquery AO dag $\mathbf{G}'$ for $\mathbf{Q}$ and $\mathbf{V}_0$, subdag of $\mathbf{G}$, such that the combined cost of evaluating the queries in $\mathbf{Q}$ according to the query evaluation plans represented in $\mathbf{G}'$, and maintaining the new materialized views represented in $\mathbf{G}'$ is minimal. In our simplified example we let the multiquery AO dag of Figure 1(a) be the input to the problem.

3.2 An Example of Application of the Rules

In order to deal with the problem we first construct a multiquery AO dag $\mathbf{G}_0$ from $\mathbf{G}$ as follows: Remove from $\mathbf{G}$ all the nodes and edges that appear solely in evaluation plans of the marked nodes (and not in those of any other query). $\mathbf{G}_0$ is called *initial multiquery AO dag.* All the marked nodes in $\mathbf{G}_0$ are sink nodes. In our example, the initial multiquery AO dag is shown in Figure 1(b). The intuition behind the construction of the initial multiquery AO dag is the following: since the marked nodes represent views that are materialized and thus are available, their evaluation need not be represented in the multiquery AO dag.

The initial multiquery AO dag is the starting point from which we can apply two transformation rules for multiquery AO dags. Consider a multiquery AO dag **G**. The first transformation rule (called here *New Materialized View Creation*)

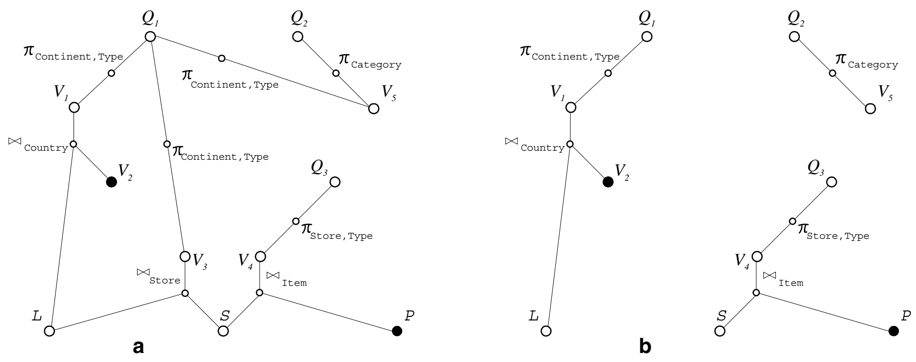

Fig. 2. A multiquery AO dag for Q_1, Q_2, and Q_3 after an application of (a) the first rule, and (b) the second rule.

can be applied to a view node V in **G** that is not a sink node and to a query node Q in **G** that can be evaluated in **G** using V. Nodes V and Q can coincide. New Materialized View Creation rule makes node V a sink node (new materialized view) and forces query Q to be evaluated using V. In order to do so, first, it removes from **G** all the nodes and the edges that are solely used for evaluating V (and not any other query). Then, it removes from **G** all the nodes and edges that are solely used for evaluating Q without using V. Figure 2(a) shows the result of applying the New Materialized View Creation rule to the view node V_5 and the query node Q_2 of the multiquery AO dag of Figure 1(b). Note that view node V_5 is now a sink node and query Q_2 can be evaluated only using V_5. The evaluation plans of the other queries are not affected.

The second transformation rule (called here *Evaluation Plan Elimination*) can be applied to a view node V in **G** that is a sink or query node and to a query node Q in **G** that can be evaluated in **G** using V but also without using V. Evaluation Plan Elimination rule forces Q to be evaluated using V. Thus, it removes from **G** all the nodes and edges that are solely used for evaluating Q without using V. Figure 2(b) shows the result of applying the Evaluation Plan Elimination rule to the view node V_2 and the query node Q_1 of the multiquery AO dag of Figure 2(a). Note that query Q_2 can now be evaluated only using V_2. The evaluation plans of the other queries are not affected.

If we assume that a new materialized view is used in at least one optimal query evaluation plan of a new query (that is, views are not materialized in the DW exclusively for supporting the view maintenance process) the previous two rules are sound and complete [29]. This means that an optimal solution to the problem (a multiquery AO dag, subdag of the initial multiquery AO dag, that yields the minimal combined cost) can be obtained by finitely applying in sequence the two transformation rules to the initial multiquery AO dag.

4 The Randomized Approach

Each solution to an optimization problem can be seen as a *state* in a *search space*. Each state has a *cost* associated with it which is given by a *cost function*. The states are connected by directed edges that are defined by moves. A *move set* is the set of moves available to go from one state to another. A move that reduce the cost of a state is called *downhill move*, while one that increases the cost of a state is called *uphill* move. Two states are called *adjacent* if one can go from one to the other by one move. A *local minimum* is a state that has the lowest cost among all its adjacent states. A *global minimum* is a state that has the lowest cost among all the states. In a search space there can be more than one local and global minimum. The optimal solution to the problem corresponds to a global minimum. Randomized algorithms perform a“random” walk through a solution space along the edges defined by the moves. Simulated Annealing is a randomized algorithm that has been successfully applied to a great number of different optimization problems. It has theoretical foundations and simplicity. We show in this section how the Simulated Annealing process can be adapted to the incremental DW design problem.

4.1 Simulated Annealing

The Simulated Annealing algorithm [16] tries to simulate the annealing process of crystals. In this process the temperature of a fluid is slowly decreased until the system reaches a state of minimum energy. The slower the temperature reduction, the lower the energy of the final state. The cost function of a state plays the role of the energy in the annealing process of crystals. As in other optimization techniques, the Simulated Annealing algorithm always accepts downhill moves but also uphill moves with some probability that depends on a number of parameters.

Figure 3 illustrates the behavior of Simulated Annealing. Two nested loops can be seen in this algorithm. In the inner loop the temperature is kept constant. A downhill move is always allowed. An uphill move is allowed with some probability that depends on the temperature and the difference between the actual state’s cost and the new state’s cost. The inner loop is finished when an equilibrium condition is met. Then the temperature is reduced and the inner loop is started again. The outer loop is finished when a freezing condition is met.

A number of parameters need to be determined for applying the algorithm to the incremental DW design problem. States, moves and cost function are problem dependent parameters. The parameters: initial state, initial temperature, temperature reduction, equilibrium condition, and freezing condition depend on the implementation of the process.

```
Input: $initial\_state$, $initial\_temperature$;
Output: $minstate$;

begin
  $minstate := initial\_state$; $cost := Cost(initial\_state)$; $mincost := cost$;
  $temp := initial\_temperature$;
  repeat
    repeat
      $newstate$ :=state after random move; $newcost := Cost(newstate)$;
      if $newcost \leq cost$ then
        $state := newstate; cost := newcost$
      else with probability $e^{\frac{newcost-cost}{temp}}$
        $state := newstate$; $cost := newcost$
      end;
      if $cost < mincost$ then
        $minstate := state$; $mincost := cost$
      end
    until equilibrium not reached;
    reduce temperature
  until not frozen;
  return $minstate$
end
```

Fig. 3. Simulated Annealing Algorithm

4.2 Problem-Dependent Parameters

Consider a multiquery AO dag $\mathbf{G}$, for a set $\mathbf{Q}$ of new queries and a set $\mathbf{V}_0$ of old views, which represents alternative evaluation plans for the queries in $\mathbf{Q}$ over the source relations.

States. A state is a multiquery AO dag $\mathbf{G}'$ for $\mathbf{Q}$ and $\mathbf{V}_0$ such that (a) $\mathbf{G}'$ is a subdag of $\mathbf{G}$, and (b) all the marked view nodes in $\mathbf{G}'$ are sink nodes. Thus, the search space is a graph of multiquery AO dags.

Moves. There is a move (edge) from a state s to a state s' if s' can be obtained from s by applying any of the two transformation rules presented in the previous section. Each of these rules removes at least one edge from the multiquery AO dag to which it is applied. Thus, the search space defined this way is an acyclic directed graph. The Simulated Annealing process requires the search space to be a strongly connected graph i.e. for any two states s and s', there is path in the search space from s to s' and from s' to s. In the implementation of the algorithm, when a random move leading to a state s is performed, the adjacent to s state is kept available. Therefore, the moves are reversible. This way the previous requirement is satisfied.

Cost function. Given a state $\mathbf{G}'$, the evaluation cost $E(\mathbf{Q})$ of the queries in $\mathbf{Q}$ is the (possibly) weighted sum of the cost of the optimal evaluation plan of each query in $\mathbf{Q}$ among the plans represented in $\mathbf{G}'$. The weights in $E(\mathbf{Q})$ reflect the frequency (or importance) of the corresponding queries. Let $\mathbf{V}$ be the set of new materialized views in $\mathbf{G}'$ (that is, the set of sink nodes that are not in $\mathbf{V}_0$).

The view maintenance cost $M(\mathbf{V})$ of the new views in $\mathbf{V}$ is the weighted sum of propagating the changes of the source relations to the views that are affected by these changes. The weights in $M(\mathbf{V})$ reflect the frequencies of propagating the changes of the corresponding source relations. The cost of the state $\mathbf{G}'$ is the combined sum $E(\mathbf{Q})+c*M(\mathbf{V})$. The parameter c indicates the importance of the query evaluation vs. the view maintenance cost and is set by the DW designer. A typical value of c is 1. $c < 1$ privileges the query evaluation cost, while $c > 1$ privileges the view maintenance cost in the incremental design of the DW. If the query evaluation cost is more important, the DW designer has the choice to give c a value much smaller than 1 in order to determine a view selection that has good query performance, and conversely. Low query evaluation cost can be obtained by materializing all the new queries (in general view nodes close to the root nodes of the graph). Low view maintenance cost can be obtained by materializing all the base relations that are not already materialized (non-marked sink nodes in the initial multiquery AO dag).

4.3 Implementation Dependent Parameters

Initial state. A theoretical analysis of the simulated annealing but also multiple experimental studies show that the effectiveness of the algorithm in finding the global minimum state is independent of the choice of the initial state [14]. Therefore we select as an initial state the initial multiquery AO dag $\mathbf{G}_0$.

Initial temperature. The initial temperature has to be sufficiently low to allow many uphill moves to be performed in the beginning. After a series of experiments we found that an initial temperature of twice the cost of the initial state shows a good performance. An increase of the multiplicative factor beyond 2 does not improve significantly the quality of the solution returned while importantly increasing the execution time of the algorithm.

Temperature reduction. Different previous works [24,13] have successfully experimented with a temperature reduction process where the new temperature is a percentage α of the old temperature and this percentage remains constant during the whole process. A typical value for α is in the range 0.9 to 0.95. Our experimental results indicated that 0.9 shows a good trade off between quality of the solution returned and execution time.

Equilibrium condition. This condition determines the number of times the inner loop of the algorithm is executed. This number can be dependent on the current temperature. Here it is assumed to be constant throughout the annealing and is experimentally set to be equal to 6.

Freezing condition. This condition determines when the annealing process terminates. The temperature is so low that accepted moves (either uphill or downhill) are negligible. A typical condition that is also adopted here consists of two tests: whether the temperature is below 1 and whether the final state does not change for four consecutive iterations through the outer loop of the algorithm.

Finally, each possible move from a state has equal probability to be chosen.

5 Experimental Results

We have performed a number of experiments to evaluate the Simulated Annealing process for incrementally designing a DW. We have chosen an input multiquery AO dag where the source relations are the dimension tables and the fact table presented in Section 3. The multiquery AO dag comprises both unary (generalized projection) and binary (natural join) operation nodes. It is reasonably sized so that an exhaustive algorithm can also be executed on it. We have adopted a simple cost model for evaluating the queries and maintaining the views. We consider that each view is maintained separately, and the materialized views are maintained by recomputation. The weights in the cost formulas are equal to 1. We have measured the time needed to execute the Simulated Annealing algorithm, and the quality of the solution returned when different factors of the problem vary. The quality of the solution is expressed as a percentage of the cost of the solution returned by the Simulated Annealing approach divided by the cost of the solution returned by the exhaustive algorithm. The varying factors are the size of the multiquery AO dag, the number of new queries, the number of marked nodes, and the parameter c in the cost function.

Experiment 1: The size of the Multiquery AO dag varies. This size is expressed as the number of view nodes in the Multiquery AO dag. Figures 4(a) and (b) plot the solution quality and the execution time against the number of view nodes in the multiquery AO dag. Each point in these and in the subsequent plots is averaged over four executions of the algorithm. The solution quality degrades as the number of view nodes increases but does not drop below 70%.

Experiment 2: The number of queries varies. Figures 4(c) and (d) plot the solution quality and the execution time against the number of query nodes in the multiquery AO dag. The solution quality increases and the execution time degrades as the number of query nodes increases. This is due to the fact that in contrast to the other view nodes, the query nodes are not removed from a multiquery AO dag by an application of the rules. Therefore, an increase in the number of the query nodes reduces the size of the search space.

Experiment 3: The number of marked nodes varies. Figures 4(e) and (f) plot the solution quality and the execution time against the number of marked nodes in the multiquery AO dag. The solution quality increases and the execution time degrades as the number of marked nodes increases. This is due to the facts that (a) the marked nodes are not removed from a multiquery AO dag by an application of the rules, and (b) their evaluation plans are removed from the multiquery AO dag. Therefore, as with the query nodes, an increase in the number of the marked nodes, reduces the size of the search space.

Experiment 4: The parameter c varies. Figures 4(g) and (h) plot the solution quality and the execution time against parameter c of the cost function. The solution quality increases and the execution time degrades as parameter c increases. This is due to the fact that when $c > 1$, the cost of a state is determined primarily by the materialized view maintenance cost. Low view maintenance cost

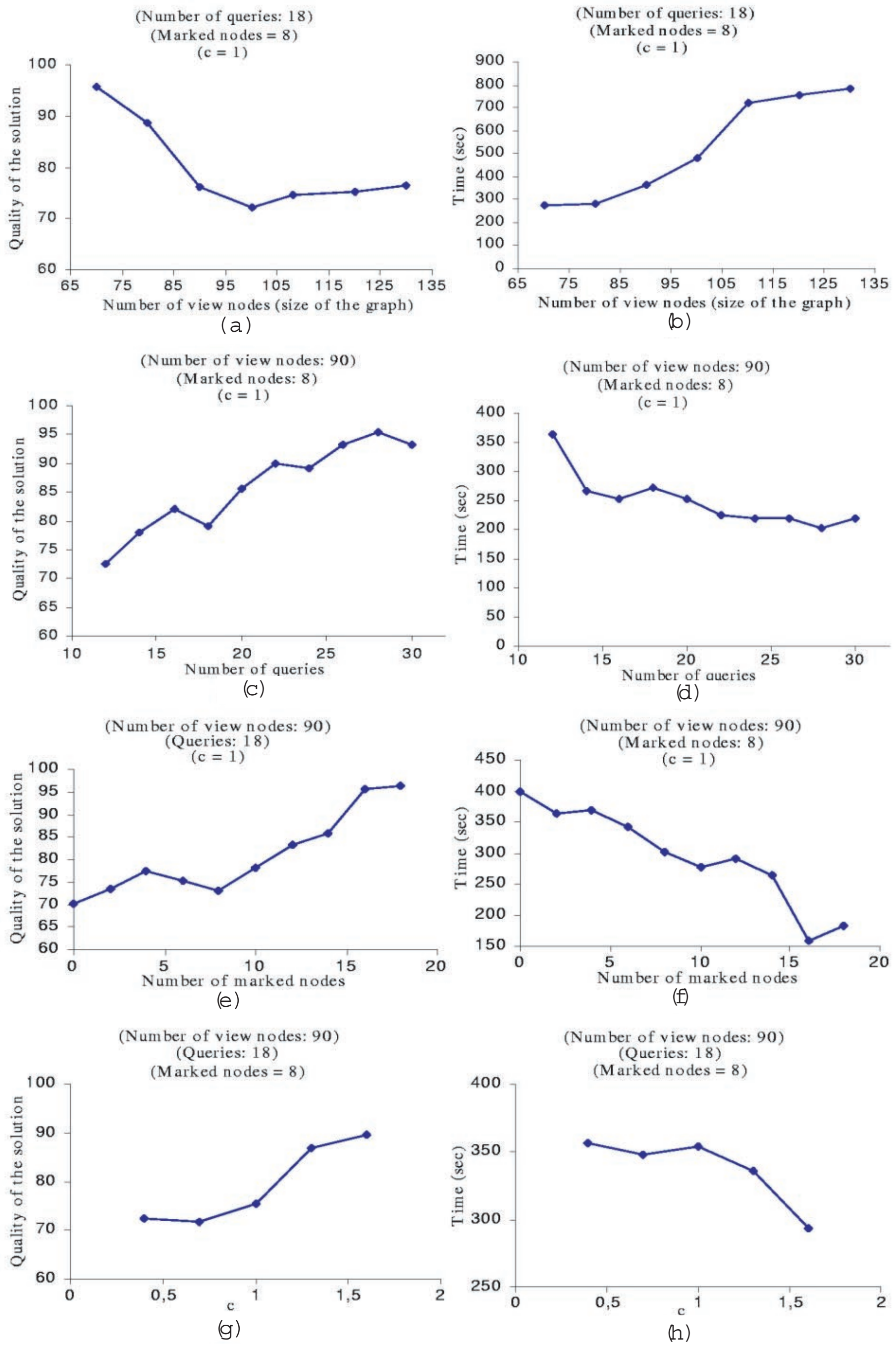

Fig. 4. Quality and time vs. number of view nodes, number of queries, number of marked view nodes and parameter *c*.

can be obtained by the states that are close to the initial state (that is, can be reached by few moves). In fact, if the cost of a state is determined uniquely by the view maintenance cost, the initial state has the minimal cost. The opposite happens when $c < 1$ where the cost of a state is determined primarily by the query evaluation cost.

6 Conclusion

Data Warehouses are dynamic entities that evolve over time. As the needs of the analysts grow new queries need to be satisfied. In general, new views need to be materialized either for satisfying the new queries or for performance reasons. Re-implementing the DW from scratch is complex and time consuming. A better solution is the incremental design of a DW where new views are materialized in order to minimize the combined evaluation cost of the new queries and the maintenance cost of the new views. A previous work on this problem provided theoretical foundations for a solution and an exhaustive algorithm. However, due to the size of the problem an exhaustive algorithm cannot be of practical interest in real cases.

We have presented a randomized approach based on the Simulated Annealing process for incrementally designing a DW. Simulated Annealing is an algorithm good for optimization problems with a large search space. The definition of the search space has been based on an AND/OR dag representation for multiple queries and views that is able to represent alternative evaluation plans of the queries from the materialized views and subexpression sharing among queries and between queries and views. Our approach has been implemented. The experimental results are encouraging and show the feasibility of the approach.

In this work we impose a constraint on a materialized view selection: the new views and the old views together must be able to answer all the new queries. An interesting extension of this work concerns the design of efficient approaches for the incremental design of a DW when additional constraints are imposed, as for instance, space or view maintenance cost constraints and integrity constraints.

References

1. E. Baralis, S. Paraboshi, and E. Teniente. Materialized view selection in a multidimensional database. In *Proc. of the 23rd VLDB Conf.*, pages 156–165, 1997.
2. S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26(1):65–74, 1997.
3. S. Cohen, W. Nutt, and A. Serebrenik. Rewriting aggregate queries using views. In *Proc. of the 18th ACM PODS Symp.*, pages 155–166, 1999.
4. S. Dar, H. V. Jagadish, A. Y. Levy, and D. Srivastava. Answering SQL Queries with Aggregation using Views. In *Proc. of the VLDB Conf.*, pages 318–329, 1996.
5. G. Graefe and W. J. McKenna. The Volcano Optimizer Generator: Extensibility and Efficient Search. In *Proc. of the 9th ICDE Conf.*, pages 209–217, 1993.
6. T. Griffin and L. Libkin. Incremental Maintenance of Views with Duplicates. In *Proc. of the ACM SIGMOD Conf.*, pages 328–339, 1995.

7. A. Gupta, V. Harinarayan, and D. Quass. Aggregate-Query Processing in Data Warehousing Environments. In *Proc. of the VLDB Conf.*, pages 358–369, 1995.
8. A. Gupta and I. S. Mumick. Maintenance of materialized views: Problems, techniques and applications. *Data Engineering*, 18(2):3–18, 1995.
9. 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.
10. H. Gupta and I. S. Mumick. Selection of Views to Materialize Under a Maintenance Cost Constraint. In *Proc. of the 7th ICDT Conf.*, pages 453–470, 1999.
11. J. Hammer, H. Garcia-Molina, J. Widom, W. Labio, and Y. Zhuge. The Stanford Data Warehousing Project. *Data Engineering*, 18(2):41–48, 1995.
12. V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing Data Cubes Efficiently. In *Proc. of the ACM SIGMOD Conf.*, 1996.
13. Y. Ioannidis and Y. Kang. Randomized algorithms for optimizing large join queries. In *Proc. of the ACM SIGMOD Conf.*, pages 9–22, 1990.
14. Y. Ioannidis and E. Wong. Query optimization by simulated annealing. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 9–22, 1987.
15. R. Kimball. *The Data Warehouse Toolkit.* John Wiley & Sons, 1996.
16. S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.
17. Y. Kotidis and N. Roussopoulos. DynaMat: A Dynamic View Management System for Data Warehouses. In *Proc. of the ACM SIGMOD Conf.*, pages 371–382, 1999.
18. A. Levy, A. O. Mendelson, Y. Sagiv, and D. Srivastava. Answering Queries using Views. In *Proc. of the ACM PODS Symp.*, pages 95–104, 1995.
19. D. Quass. Maintenance Expressions for Views with Aggregation. In *Workshop on Materialized Views: Techniques and Applications*, pages 110–118, 1996.
20. D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making Views Self Maintainable for Data Warehousing. In *Proc. of the 4th PDIS Conf.*, pages 158–169, 1996.
21. 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 Conf.*, pages 447–458, 1996.
22. M. Steinbrunn, G. Moerkotte, and A. Kemper. Heuristic and randomized optimization for the join ordering problem. *VLDB Journal*, 6:191–208, 1997.
23. A. Swami. Optimization of Large Join Queries: Combining Heuristics and Combinatorial Techniques. In *Proc. of the ACM SIGMOD Conf.*, pages 367–376, 1989.
24. A. Swami and A. Gupta. Optimization of Large Join Queries. In *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 8–17, 1988.
25. D. Theodoratos. Detecting Redundant Materialized Views in Data Warehouse Evolution. *Information Systems*, 26(5), 2001.
26. D. Theodoratos, S. Ligoudistianos, and T. Sellis. View Selection for Designing the Global Data Warehouse. To appear in *Data and Knowledge Engineering.*
27. D. Theodoratos and T. Sellis. Data Warehouse Configuration. In *Proc. of the 23rd Intl. Conf. on Very Large Data Bases*, pages 126–135, 1997.
28. D. Theodoratos and T. Sellis. Designing Data Warehouses. *Data and Knowledge Engineering, Elsevier*, 31(3):279–301, Oct. 1999.
29. D. Theodoratos and T. Sellis. Incremental Design of a Data Warehouse. *Journal of Intelligent Information Systems, Kluwer Academic Publishers*, 15(1):7–27, 2000.
30. J. Widom. Research Problems in Data Warehousing. In *Proc. of the 4th Intl. Conf. on Information and Knowledge Management*, pages 25–30, Nov. 1995.
31. J. Yang, K. Karlapalem, and Q. Li. Algorithms for Materialized View Design in Data Warehousing Environment. In *Proc. of the VLDB Conf.*, pages 136–145, 1997.