ELSEVIER

Data & Knowledge Engineering 37 (2001) 203–216

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

Materialized view selection under the maintenance time constraint

Weifa Liang [a,*], Hui Wang [b], Maria E. Orlowska [b]

[a] *Department of Computer Science, Australian National University, Canberra, ACT 0200, Australia*
[b] *School of Computer Science and Electrical Engineering, University of Queensland, St. Lucia, Qld 4072, Australia*

Received 26 June 2000; received in revised form 12 October 2000; accepted 4 January 2001

Abstract

A data warehouse is a data repository which collects and maintains a large amount of data from multiple distributed, autonomous and possibly heterogeneous data sources. Often the data is stored in the form of materialized views in order to provide fast access to the integrated data. One of the most important decisions in designing a data warehouse is the selection of views for materialization. The objective is to select an appropriate set of views that minimizes the total query response time with the constraint that the total maintenance time for these materialized views is within a given bound. This view selection problem is totally different from the view selection problem under the disk space constraint. In this paper the view selection problem under the maintenance time constraint is investigated. Two efficient, heuristic algorithms for the problem are proposed. The key to devising the proposed algorithms is to define good heuristic functions and to reduce the problem to some well-solved optimization problems. As a result, an approximate solution of the known optimization problem will give a feasible solution of the original problem.

Keywords: Incremental maintenance; Design of data warehouse; Materialized view selection; Maintenance time constraint; Data integration; Heuristic algorithms

1. Introduction

A data warehouse is a repository of integrated information available for query and analysis. One of the major motivations for constructing data warehouses is for queries that can be answered using the information stored there without a need to be translated and shipped to remote sources for execution. Also, the warehouse data is always available for users, even when the remote sources are not accessible at the times of the local source's maintenance period. Often the data in the data warehouse is stored in a form of materialized views in order to accelerate on-line analytical processing (OLAP).

* Corresponding author. Tel.: +61-2-6125-3019; fax: +61-2-6125-0010.
E-mail addresses: wliang@cs.anu.edu.au (W. Liang), hwang@csee.uq.edu.au (H. Wang), maria@csee.uq.edu.au (M.E. Orlowska).

0169-023X/01/$ - see front matter
PII: S0169-023X(01)00007-6

1.1. Some related work

The selection of views for materialization is one of the most important issues in the design of data warehouses. Given the disk space constraint, substantial effort in selecting a set of views to materialize with different optimization objectives has been taken over the past several years [1,3–5,7,11]. For example, Harinarayan et al. [7] provide an efficient, polynomial greedy algorithm to select views for materialization in order to minimize the total query response time for a data cube that delivers a nearly optimal solution. Gupta et al. [5] extend the results to the selection of views and indices in data cubes. Baralis et al. [1], Gupta [4], Ross et al. [9] and Yang et al. [11] present algorithms for the selection of materialized views by taking into account the view maintenance cost and aim to minimize both the total query response time and the view maintenance overhead under the given disk space constraint. Theodoratos and Sellis [10] reformulate the view selection problem as a state space optimization problem by providing various optimization frameworks and heuristics.

It is obvious that, in a data warehouse, its query performance will be improved tremendously as more and more views are materialized. With the ratio $$/disk volume constantly dropping, the disk storage constraint is no longer the limiting factor of the materialized view selection. The view maintenance time window is limited (usually the data warehouse maintenance is carried out at night or over weekends) due to too many materialized views in the data warehouse which need to be maintained. More view materialization implies that a larger time maintenance window is needed. Therefore, the view maintenance window is a critical parameter in the design of a data warehouse, which constrains the number of views to be materialized, and thereby determines the scalability and the functionality of the data warehouse in the end.

Recently Gupta and Mumick [6], and Kotidis and Roussopoulos [8] first explored the view selection problem under a given maintenance time constraint. Kotidis and Roussopoulos [8] considered the data cube case where a horizontal fragment of a cubeid of the data cube is stored as a materialized view. Here a cubeid is an aggregate table derived from the fact table. Thus, a cubeid can be partitioned into several fragments and the fragments are stored as materialized views. To answer a user's aggregate query, it is obvious that several materialized views are often required. To this end, they suggest a heuristic approach for selecting fragments to materialize. Gupta and Mumick [6] introduce a theoretical framework by modeling the view selection problem under the maintenance time constraint as an OR-graph for the case of data cubes, and an AND–OR graph for a more general setting, with presented heuristic algorithms for the problem. Their algorithm for the OR-graph, however, is less efficient than a naive algorithm which is shown as follows. For the OR-graph, they adopted a greedy strategy. That is, the solution is built incrementally. Initially, the set of materialized views $\mathrm{MV} = \emptyset$. Then, the algorithm repeats the following step until the total maintenance time for the views in MV is beyond the given maintenance time limit. At each step all possible inverted trees in the OR-graph are considered and one of the inverted trees T is chosen such that: (i) T has the largest benefit; and (ii) the maintenance time for views in $\mathrm{MV} \cup V(T)$ is within the given maintenance time limit, where $V(T)$ is the view set in T. Let m be the number of vertices in the OR-graph, then the time complexity of their algorithm is

$$\Omega\left(\sum_{k=1}^{m-1}\binom{m}{k}\right) = \Omega(2^m)$$

because there are $\Omega(2^m)$ possible inverted trees in the graph. There is a simple naive algorithm for finding the set of materialized views as follows. For each different k, $1 \leqslant k < m$, choose k views as a set of potential materialized views from the m views. There are $\binom{m}{k}$ such sets for a given k. Once a set of k views satisfies that (i) all the queries can be answered using the views in the set, and (ii) the total maintenance time for the views in the set is within the given maintenance time limit, that view set will become a candidate of the feasible solution of the problem. Finally, one feasible solution leading to the minimum query response time is chosen from these possible feasible solutions. Thus, the time complexity of the naive algorithm is

$$O\left(\sum_{k=1}^{m-1}\binom{m}{k}\right) = O(2^m),$$

which is no worse than the one in [6].

Although there are some similarities between the view selection problems under the disk space constraint and under the maintenance time constraint, they are, in fact, significantly different, which is explained as follows. For the space constraint version, the optimization objective is the total amount of space used for the view maintenance, while the total disk space occupied by a set of views always increases when more views are materialized. This is the so-called *monotonicity* of the optimization function. However, under the maintenance time constraint, the optimization objective is the total amount of maintenance time spent for a set of materialized views, while this optimization objective may not have the monotonicity property. In some cases, it is possible that the maintenance time for a set of views will decrease when more views are materialized. This non-monotonic nature of the maintenance time increases the complexity of designing efficient algorithms for the view selection problem under the maintenance time constraint. It is based on the above discussions. In this paper we dedicate ourselves to develop efficient algorithms for the view selection problem under the maintenance time constraint.

1.2. Our contributions

In this paper two efficient heuristic algorithms for the view selection problem under the maintenance time constraint are proposed. The two-phase algorithm consists of two phases. It optimizes the total query response time in the first phase, and chooses views for materialization under the given maintenance time constraint in the second phase. Several possible improvements to this algorithm are also discussed. The integrated algorithm takes into account both the maintenance time and the query response time simultaneously. So, the solution it delivers is better than that delivered by the two-phase algorithm as well as its variant of the two-phase algorithm. The integrated algorithm, however, takes a longer time to obtain a solution.

The key to devising the proposed algorithms is to define good heuristic functions and to reduce the problem to some well-solved optimization problems. Accordingly, any approximate solution of the known optimization problem will give a feasible solution of the problem under discussion.

Although both proposed algorithms are based on an assumption of the static setting in which the number of views and queries are stabilized and fixed up during the selection of views for materialization, these algorithms can easily be adopted in a dynamic setting through minor modifications. In the dynamic environment, whether or not a view is in the set of materialized views can be changed dynamically, and so too can a query in the query set.

1.3. Structure of the paper

The rest of the paper is organized as follows. Section 2 defines the problem precisely. Section 3 presents the two-phase algorithm and its variant. Section 4 proposes the integrated algorithm using different heuristics. Section 5 concludes the paper.

2. Preliminaries

Following [6], in this paper we assume that the views in the data warehouse form a directed graph $G(V, E)$ which is an OR-graph DAG, where V is the set of views and E is the set of directed edges. A directed edge $\langle u, v\rangle \in E$ from u to v implies that vertex (view) u can be derived from vertex (view) v. Many practical applications can be modeled as an OR-graph including the data cube. For example, Fig. 1 shows such an OR-graph. View x can be derived from u, while view y can be derived from either u or v.

The view selection problem under the maintenance time constraint can be formulated as follows. Given a maintenance time window T and n queries $q_1, q_2, \ldots, q_n$ with the query frequency f_i of q_i, $1 \leqslant i \leqslant n$, assume that each query can be answered using at least one of the views in V. The problem is to find a subset S ($\subset V$) of views for materialization such that: (i) all the queries can be answered using the materialized views; (ii) the total query response time is minimized; and (iii) the total maintenance time for these materialized views is within T.

3. A two-phase view selection algorithm

In this section we propose a heuristic algorithm for finding a feasible solution for the problem. The proposed algorithm consists of two phases. In the first phase it finds a set S ($\subset V$) of views such that the total query response time is minimized. If the maintenance time for the views in S is also within the given maintenance time bound, then S is the solution of the problem, and it has

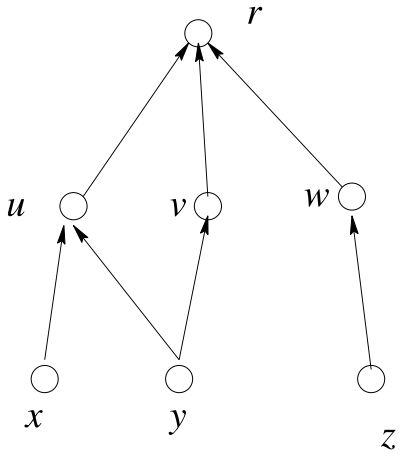

Fig. 1. An OR-graph.

been done. Otherwise, in the second phase, a subset S' ($S' \subset S$) of S will be chosen such that the total maintenance time for the views in S' is kept within the maintenance time bound.

The algorithm is a greedy algorithm, in which a view is picked up and materialized, solely based on its gain benefit to the data warehouse at that moment, and where the gain benefit function on the views will be defined later.

3.1. The maintenance time of a materialized view

Let $S(\subset V)$ be the set of materialized views in the data warehouse. Assume that each materialized view v has another materialized view $p(v)$ as its *parent view*, which means v can be derived from $p(v)$ through grouping on some attributes in $p(v)$.

To respond to a source update which effects v, the maintenance time T_v associated with v, therefore, is

$$T_v = t(p(v), v) + t(v, v), \tag{1}$$

where $t(p(v), v)$ is the update cost of v due to an update applied to its parent view $p(v)$ (e.g., consider an insertion case, there is a $\Delta p(v)$ which propagates to v) and $t(v, v)$ is the update cost of v depending on the size of v. The idea behind formula (1) can be explained as follows. Assume that a view v with size of 10 000 can be derived from two views s and t, respectively. When there is an update in a source which leads to incremental updates δs with size of 100 to s and δt with size of 50 to t. Thus, if v had chosen $s(t)$ as its parent view, then, the maintenance time of v needed is a linear function of its size of 10 000 and the size of the incremental update δs with 50 (δt with 100).

By the above assumption, each materialized view in the data warehouse has a parent view from which the view is derived, and all materialized views in the data warehouse are organized by a forest of trees. Fig. 2 shows a set of materialized views in a data warehouse, where view v chooses view u as its parent view. Thus, any source update will be propagated to the views at the tree roots r_1, r_2 and r_3 first. The incremental update to v can be committed after its parent view u propagates the corresponding incremental update δu to it.

3.2. The algorithm

The basic idea behind our algorithm is as follows. We first select a subset S from the view set V such that the total query response time to answer the n queries using the views in S is minimized. If the total maintenance time $t_{\text{maintenance}}(S)$ $(= \sum_{v \in S} T_v)$ for the views in S satisfies $t_{\text{maintenance}}(S) \leqslant T$,

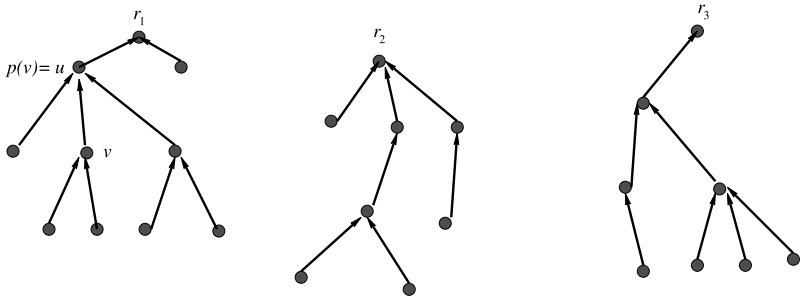

Fig. 2. The organization of materialized views in a data warehouse.

then, all the views in S can be materialized, and S is a solution of the problem. However, usually there is no such S that $t_{\text{maintenance}}(S) \leqslant T$. Instead, a subset S' of S can be found, and such an S' must satisfy the following conditions: (i) $S' \subset S$; (ii) all the views in $S - S'$ can be derived from S', and thus all the queries can be answered using the views in S', because each view in S' is a representative of one or multiple views in S; (iii) $T_{\text{maintenance}}(S') \leqslant T$; and (iv) the total query response time is minimized. A representative q of a set of views is such a view that every other view in the set can be derived from the view. In the following we focus on finding S and S' if needed, in more details.

Our approach is to reduce the view selection problem under the maintenance time constraint to a minimum weighted maximum cardinality matching problem on an auxiliary weighted, bipartite graph H which will be defined later. This latter problem is a well-known one which can be solved in polynomial time. Thus, a feasible subset $S(\subseteq V)$ for the problem is obtained by transforming the solution of this latter problem.

To this end, an auxiliary weighted bipartite graph $G_B = (X, Y, E_{XY}, \omega_1)$ is defined as follows. X is the view set ($X = V$) with $|V| = m$ and Y is the query set with $|Y| = n$, i.e., $Y = Q = \{q_1, q_2, \ldots, q_n\}$. There is an edge $(v_j, q_i) \in E_{XY}$ between $v_j \in X$ and $q_i \in Y$ if q_i can be answered using v_j, $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$. The weight $\omega_1(v_j, q_i)$ associated with the edge (v_j, q_i) is the weighted cost of answering q_i using view v_j, which is defined as follows:

$$\omega_1(v_j, q_i) = \text{size}(v_j) c(v_j, q_i) f_i, \tag{2}$$

where $size(v_j)$ is size of view v_j, $c(v_j, q_i)$ is a constant which is related to the processing cost by filtering a tuple in v_j to see whether there is a derived tuple in q_i, and f_i is the query frequency of query $q_i \in Q$, $1 \leqslant i \leqslant n$ and $1 \leqslant j \leqslant m$.

The original optimization objective S can be stated equivalently as follows. The objective is to find a subset $X' \subseteq X$ in G_B such that: (i) every vertex $y \in Y$ chooses a vertex $v(y) \in X'$; and (ii) $\sum_{y \in Y} \omega_1(v(y), y)$ is minimized. Condition (i) says that every query can be answered, using a view in X'; condition (ii) says that the total query response time is minimized.

Clearly, $\sum_{y \in Y} \omega_1(v(y), y)$ is a lower bound of the actual total query response time, because we assume here that all the views in $X'(= S)$ are materialized and the maintenance time for them is no greater than T. Such X' can be found in polynomial time by reducing the problem on G_B to a minimum weighted, maximum cardinality matching problem on another auxiliary weighted, bipartite graph $H = (X_H, Y_H, E_H, \omega_2)$, which is defined as follows. Let $d(x)$ be the vertex degree of $x \in X$ in G_B. $X_H = X \cup \{x_1, x_2, \ldots, x_{d(x)-1} \,|\, x \in X, d(x) > 1\}$, $Y_H = Y$. $E_H = E_{XY} \cup \{(x_i, y) \,|\, (x, y) \in E_{XY}, 1 \leqslant i \leqslant d(x) - 1\}$. The weights associated with the edges in H are defined as $\omega_2(x, y) = \omega_1(x, y)$ for every $(x, y) \in E_{XY}$ and $\omega_2(x_i, y) = \omega_1(x, y)$ for every $(x_i, y) \in E_H$, $1 \leqslant i \leqslant d(x) - 1$.

Let $M(\subseteq E_H)$ that is an edge subset be a minimum weighted, maximum cardinality matching in H, and let $V(M)$ be the endpoint set of the edges in M. Define $V_X(M) = V(M) \cap X_H$, which is a subset of X_H. Map the vertices in $V_X(M)$ back to the vertices in G_B, i.e., a vertex x_i in X_H of H will be mapped to its original vertex x in X of G_B. As results, these original vertices form the vertex set S, which is what we need.

It is well known that the minimum weighted, maximum cardinality matching problem can be solved in polynomial time [2]. Having the matching M in H, each query q knows from which view it can be answered. Without loss of generality, let $(p'(q), q) \in M$ and $p'(q) \in X_H$ be mapped back to $p(q) \in X$, then, $p(q) \in S$ is the view from which query $q \in Q$ is answered.

We now assume that $t_{\text{maintenance}}(S) > T$. Otherwise, S is the solution of the problem. We need to find $S' \subset S$ that satisfies the above constraints defined. Before we proceed, a *directed* subgraph $G_1 = (S, E_1)$ is constructed, where $\langle u, v\rangle \in E_1$ if $v \in S$ and $u \in S$ and u can be derived from v. Associated with each view v in S, there is a weight $w(v) = \sum_{j=1}^{k} f_{i_j}$ where f_{i_j} is the query frequency of q_{i_j} and q_{i_j} has chosen v in M, i.e., $(p(q_{i_j}), v) \in M$, assuming that $1 \leqslant i_j \leqslant k \leqslant |S|$.

By the initial assumptions that each materialized view has a materialized parent view from which the view is derived and its incremental update is received, and every virtual view in $S - S'$ can be derived from one of the materialized views, the rest is dedicated to find S'. Our approach to finding S' is a greedy approach.

The algorithm proceeds as follows. Initially, it sets $S' = \emptyset$. It then repeats the following step as long as $t_{\text{maintenance}}(S') < T$ and there exists a $v \in S - S'$ such that $t_{\text{maintenance}}(S' \cup \{v\}) \leqslant T$. Each time a virtual view $v \in S - S'$ is picked up for materialization if v has the maximum *gain benefit* $g(v)$ to the data warehouse at the current step, where $g(v)$ is defined below.

Let ΔT_v be the net increase in the total maintenance time due to the materialization of v. Then, $\Delta T_v = t_{\text{maintenance}}(S' \cup \{v\}) - t_{\text{maintenance}}(S')$, which is computed as follows. Assume that $v_1, v_2, \ldots, v_l$ are those materialized views that will become the children of v after v is materialized. Let T'_{v_i} and T_{v_i} be the maintenance times of v_i before and after the materialization of v. Then, $T_{v_i} < T'_{v_i}$. Otherwise, v_i would not change its parent view to v if there is no improvement in terms of its maintenance time denoted by $\nabla_{v_i} = T'_{v_i} - T_{v_i}$. Then,

$$\Delta T_v = T_v - \sum_{i=1}^{l} \nabla_{v_i}. \tag{3}$$

Note that, in most cases ΔT_v is non-negative. However, if $\Delta T_v < 0$, then, this means that the materialization of v does not increase but reduces the total maintenance time. Let $\text{Cost}(Q, S'')$ be the total query response time for answering queries in Q when the views in S'' are materialized. We then have the following lemma.

Lemma 1. *If* $\Delta T_v < 0$ *and* $v \notin S'$, *then* $T_{\text{maintenance}}(S') > T_{\text{maintenance}}(S' \cup \{v\})$ *and* $\text{Cost}(Q, S') \geqslant \text{Cost}(Q, S' \cup \{v\})$.

Proof. Note that $\Delta T_v = t_{\text{maintenance}}(S' \cup \{v\}) - t_{\text{maintenance}}(S')$. If $\Delta T_v < 0$, $T_{\text{maintenance}}(S') > T_{\text{maintenance}}(S' \cup \{v\})$. By the definition, $\text{Cost}(Q, S') = \sum_{q \in Q} \text{size}(p(q)) c(p(q), q) f_q$ where the materialized view $p(q)$ is the parent view of q, $c(p(q), q)$ is a constant which is related to the processing cost by filtering tuples of $p(v)$ in order to obtain q, and f_q is the query frequency of q. Then, $\text{Cost}(Q, S') - \text{Cost}(Q, S' \cup \{v\}) = \sum_{p(q)=v_i, 1 \leqslant i \leqslant l} (\text{size}(v_i) c(v_i, q) - \text{size}(v) c(v, q)) f_q \geqslant 0$, since $(\text{size}(v_i) c(v_i, q) - \textit{size}(v) c(v, q)) f_q \geqslant 0$. Thus, $\text{Cost}(Q, S') \geqslant \text{Cost}(Q, S' \cup \{v\})$.

From now on, we assume that ΔT_v is positive. Due to the materialization of v, the gain benefit of v to the data warehouse is defined as follows.

$$g(v) = \frac{\sum_{\exists v_j \in S - S'} [\textit{size}(p(v_j)) c(p(v_j), v_j) - \textit{size}(v) c(v, v_j)] w(v_j)}{\Delta T_v}, \tag{4}$$

where $c(p(v_j), v_j)$ is a constant which is related to the processing cost by filtering a tuple of $p(v_j)$ to see whether there is a derived tuple in v_j and $c(v_i, v_i) = 1$, and $size(p(v_j))c(p(v_j), v_j)$ is the time used to derive v_j from a materialized view $p(v_j)$.

Note that in expression (4) only those materialized views v_j which have changed their parents to v will bring positive benefits to the data warehouse, because the total maintenance time for all materialized views becomes smaller due to the materialization of v. Also, ΔT_v instead of T_v used in expression (4) is based on the following important observation.

When a view v is materialized, it may reduce the maintenance time of other materialized views, although it incurs $T_v = t(p(v), v) + t(v, v)$ maintenance time for its own maintenance. For example, assume that there are two materialized views A and B, and A is a parent view of B, then the maintenance time of B depends on A to some extent. Let T'_B (T'_A) and $T_B(T_A)$ be the maintenance times of $B(A)$ before and after materializing a view C, respectively. Now, assume C is materialized, and C has chosen A as its parent view, and B has chosen C as its parent view due to that $T_B < T'_B$. Thus, the *net* increase of the total maintenance time for A, B and C is $(T_A + T_B + T_C) - (T'_A + T'_B) = (T_B - T'_B) + T_C < T_C$, where $T_A = T'_A$. In this example, the materialization of C incurs T_C maintenance time for view C itself but the net increase in the total maintenance time is $T_C + (T_B - T'_B)$ $(< T_C)$, which is less than T_C.

The procedure of choosing a virtual view v for materialization based on the gain benefit of v can proceed until the total maintenance time is beyond the given bound T. Once the S' has been found, the total query response time for the n queries is then determined, which is $C_1 + C_2$, where C_1 is the weighted sum of the edges in M in the first phase, and $C_2 = \sum_{v_j \in S - S'} \text{size}(p(v_j))$ $c(p(v_j), v_j)w(v_j) + \sum_{v_j \in S'} \text{size}(v_j)w(v_j)$, is the cost to generate virtual views in $S - S'$ in the second stage. The following is the pseudo-code of the proposed algorithm.

```
Procedure   Two_Phase_Select_Views(V, Q, T)
 1.         S = ∅; /* the materialized view set */
 2.         Construct the weighted bipartite graph G_B;
 3.         Construct the weighted bipartite graph H;
 4.         Find a minimum weighted maximum cardinality matching M in H;
 5.         Let V(M) be the vertex set of M in X_H
                and V_{G_B}(M) be its corresponding vertex set of X in G_B;
 6.         S = V_{G_B}(M);
 7.         if T_maintenance(S) ⩽ T then
                materialize all views in S, exit
 8.         else
 9.             Construct a vertex-weighted directed graph G_1(S, E_1);
10.             S' = ∅;
11.             repeat
12.                 pick a v ∈ S − S' such that g(v) = max_{u∈S−S'}{g(u)}
                        and T_maintenance(S' ∪ {v}) ⩽ T
13.                 S' = S' ∪ {v};
14.                 S = S − {v};
15.             until T_maintenance(S' ∪ {v}) > T
                endif;
```

We now analyze the time complexity of the two-phase algorithm. The constructions of G_B and H take $O(mn)$ time since $|V| = m$ and $|Q| = n$. Finding a minimum weighted maximum cardinality matching M takes $O(mn^{3/2})$ time [2]. The construction of G_1 takes $O(m^2)$ time. The execution of the `repeat` loop takes $O(m^2)$ time. Therefore, the two-phase algorithm takes $O(m^2 + mn^{3/2})$ time.

3.3. Improvements of the two-phase algorithm

The above proposed algorithm gives a feasible solution for the problem with an assumption that all the views in $S - S'$ can be derived from the views in S'. Also, once a view has been chosen and materialized, it cannot be changed back to be virtual again. To overcome the drawback and remove the assumption, several variants of the proposed algorithm are presented as follows.

First, we show how to allow a view to change its status (materialized vs virtual). Consider two materialized views v and v' in S'. Assume that v is materialized prior to v'. At the time of v being materialized, the gain benefit of v to the data warehouse is larger than that of v', by our strategy of picking a view to materialize. But, with more and more views having been materialized after the materialization of v, the gain benefit contributed by v to the data warehouse may become smaller and smaller, due to the fact that some virtual views that originally had chosen v as their parent views now change their parent views to the other materialized views which were materialized after v. So, one possible modification to the two-phase algorithm is as follows. When there is no enough available maintenance time left to proceed further view materialization, a materialized view u with the minimum *loss benefit* $l(u)$ will be chosen and removed from S'. As a result, u releases a certain amounts of maintenance time, which allows the materialization procedure to continue. The loss benefit $l(u)$ of a materialized view u is defined as follows.

Let $v_1, v_2, \ldots, v_l$ be the materialized views which are the children of u. Those views will need changing their parent views after the removal of u. Let T'_{v_i} and T_{v_i} be the maintenance time of v_i before and after the removal of u. Clearly, $T_{v_i} \geqslant T'_{v_i}$. Let $\Delta_{v_i} = T_{v_i} - T'_{v_i}$, then $\Delta_{v_i} > 0$. The total maintenance time released by u, therefore, is

$$\nabla T_u = T_u - \sum_{i=1}^{l} \Delta_{v_i}, \tag{5}$$

due to the removal of u. If $\nabla T_u \leqslant 0$, this means $T_{\text{maintenance}}(S' - \{u\}) > T_{\text{maintenance}}(S')$, then whether u should be removed is determined by a random probability. In the rest of the discussion of this section, we assume $\nabla T_u > 0$. The loss benefit of u to the data warehouse then is

$$l(u) = \frac{\sum_{v \in S - S'}^{p(v)=u} [\text{size}(p'(v))c(p'(v), v) - \text{size}(v)c(u, v)]w(v) + \text{size}(u)w(u)}{\nabla T_u}, \tag{6}$$

where $p'(v)$ is the parent of v after the removal of u from S'.

Thus, a virtual view in $S - S'$ can be chosen for materialization through the removal of a minimum loss benefit materialized view from the set of materialized views. This procedure continues until the threshold of the loss benefit of the chosen materialized view is greater than a given value, or no better total query response time could be achieved after a certain number rounds of tries.

Second, we discuss here how to choose a dynamic set S, i.e., the element in S can be added or removed dynamically. The two-phase algorithm does deliver a feasible solution, but it always

chooses a subset S' from a set S to materialize when S is found. Thus, it may deliver a local optimal solution in the best possible way when S is fixed up. However, if S is not properly chosen initially, then there is no way to get a better solution for the problem no matter how much effort would be taken afterwards. Therefore, an improvement to the proposed algorithm is enabling it to search a larger view space (not only just the views in S but also the views in $V - S$) to find "better" views to materialize, and thereby find a better solution for the problem.

Assume that the view set S has been found by applying the two-phase algorithm. Define a *compatible* relation between a materialized view x and a virtual view y where $x \in S$ and $y \in V'$ ($V' = V - S$). We say that x and y are compatible if and only if $Q(x) \subseteq Q(y)$, where $Q(z)$ ($\subseteq Q$) is the query set in which each query is answerable using the materialized view z. In other words, if y is materialized and x is de-materialized at the same time, then, all those materialized views chosen x as their ancestors can now choose y as their ancestors (these views can be derived from y). The improved algorithm proceeds as follows. We first find a feasible solution S', using the proposed approach. We then improve the solution further by replacing a materialized view $v \in S'$ having a minimum loss benefit with another compatible view $u \in V'$ such that: (i) u has never been materialized; (ii) the gain benefit of u is the maximum; and (iii) the total maintenance time for the materialized views is within T, i.e., $t_{\text{maintenance}}(S' - \{v\} \cup \{u\}) \leqslant T$. This procedure can be repeated until there is no further improvement in terms of the solution obtained after a certain number of tries.

Finally, we give another approach to generate the initial set S. To find a subset S from V, the two-phase algorithm is to reduce the problem to a minimum weighted, maximum cardinality matching problem. Here, another method is suggested to generate the S. Recall that the relationships among the views in the data warehouse form an OR-graph $G(V, E)$, which is a DAG. Let $G_t(V, E^*)$ be the transitive closure of $G(V, E)$. A graph G_t is a transitive closure of another graph G if there is a directed path from u to v in G, then there is a directed edge $\langle u, v\rangle$ from u to v in G_t. Associated with each edge $\langle u, v\rangle \in E^*$, there is a weight $size(v)c(v, u)$ on it, which is the amount of time used to derive u from v.

We now define another DAG $G_{VQ}(V_{VQ}, E_{VQ})$ as follows. $V_{VQ} = V \cup V_q$ is a set of vertices, where V_q is a set of query vertices and V is a set of view vertices. A directed edge $\langle u, v\rangle \in E_{VQ}$ from u to v if and only if either (i) $\langle u, v\rangle \in E^*$ in G_t and the weight associated with it is $size(v)c(v, u)$ or (ii) $u \in V_q$ is a query and $v \in V$ is a view and q can be answered using v. The weight associated with this edge is $size(v)c(v, u)f_u$, where f_u is the query frequency of u. It is obvious that G_{VQ} is a directed, weighted graph. There is neither a directed edge between two query vertices nor a directed edge from a view vertex to a query vertex in G_{VQ}.

Having constructed G_{VQ}, add an *artificial vertex* s to it, $s \notin V \cup V_q$, and add a directed edge $\langle s, v\rangle$ to it with weight zero for all the views v that are derived from sources solely. An approximate directed Steiner tree rooted at s including all the vertices in V_q ($= Q$) on the augmented G_{VQ}, denoted by G_{VQ}^{aug}, is then found, by applying the algorithm due to Zelikovsky [12]. Remove the tree root s and all its adjacent edges from the tree, the tree then becomes a forest. Let F be the resulting forest of trees by removing all the leaf vertices (query vertices) of the approximate directed Steiner trees. As a result, the remaining vertex set of trees in F is the S. Having the initial set S, if needed, the two-phase algorithm and the above improvements can be applied to find a subset of views S' ($\subset S$) for materialization, which is a feasible solution for the problem.

4. An integrated algorithm

Although the two-phase algorithm and its variants in the preceding section give feasible solutions for the problem, they do not take into account both of the optimization objectives (the maintenance time and the query response time) simultaneously. Here an algorithm called *integrated algorithm* is presented, which will take these two optimization objectives into consideration at the same time.

The proposed approach is still a greedy approach. Initially, assume all the other views are virtual except those views which are directly derived from sources that are materialized. To answer all queries, the total query response time is very large initially if only those materialized views are used. The strategy to reduce the query response time is to materialize more and more views as long as the total maintenance time is bounded by T. A virtual view is chosen for materialization if the amount of increase in the maintenance time will bring the maximum gain benefit by the view to the data warehouse.

Suppose that S' is the set of materialized views so far and $T_{\text{maintenance}}(S')$ is the total maintenance time for the views in S'. Recall that $\text{Cost}(Q, S')$ is the total query response time for answering all the queries in Q. Consider a view $v \in V - S'$ to be materialized. The net increase in the total maintenance time is $\Delta T_v = T_{\text{maintenance}}(S' \cup \{v\}) - T_{\text{maintenance}}(S')$, and the amount of query response time reduction is $\text{COST}_v = \text{Cost}(Q, S') - \text{Cost}(Q, S' \cup \{v\})$, due to the materialization of v. Thus, each time a view $v \notin S'$ is chosen to materialize if the gain benefit $g(v)$ of v is the maximum one, and $g(v)$ is defined as follows.

$$g(v) = \frac{\sum_{q \in Q} f_q \text{Cost}(\{q\}, S') - \sum_{q \in Q} f_q \text{Cost}(\{q\}, S' \cup \{v\})}{\Delta T_v}, \tag{7}$$

where $\text{Cost}(\{q\}, S')$ is the response time of answering query q using the materialized views in S', which can be defined as follows. Recall that the weighted directed graph G_{VQ} defined in preceding section is a DAG. Associated with each edge $\langle u, v \rangle$ in G_{VQ}, the weight is either $size(v)c(v, u)$ which is the time to derive view u from view v or $size(v)c(v, u)f_u$ which is the time to derive query u from view v.

Let $\text{Length}(q, v)$ be the length of the shortest path from q to v in G_{VQ} with $v \in S'$ in terms of the weighted sum of the edges in the path. Then,

$$\text{Cost}(\{q\}, S') = \min_{v \in S'} \{\text{Length}(q, v)\}, \tag{8}$$

which is the time used for answering query q using a materialized view v'. Thus, the integrated algorithm is described below.

```
Procedure   Select_Views(V, Q, T)
1.          S' = {v : v is derived from the sources directly};
                /* the materialized view set */
2.          T' = T − T_maintenance(S'); /* the remaining maintenance time */
3.          Cost(Q, S') = Σ_{q∈Q} Cost({q}, S'); /* the query response time */
            let p(q) ∈ S' be the view from which query q is answered
                        with the minimum cost;
```

4. **while** $T' > 0$ **do**
/* Choose a view $v_0 \in V - S'$ with the maximum gain benefit*/
5. $gain\_benefit = 0$;
/* which view will bring the maximum gain benefit */
6. **for** each $v \in V - S'$ **do**
7. **for** each $q \in Q$ **do**
8. $cost(q) = Length(q, p(q))$;
endfor;
9. find a single source shortest path tree rooted at $v \in V$ in G_t;
10. **for** each $q \in Q$ **do**
11. **if** $Length(q, v) < cost(q)$ **then**
12. $cost(q) = Length(q, v)$
13. $source(q) = v$
/* v is the potential view from which q can be answered. */
endif;
endfor;
14. $\mathrm{Cost}(Q, S' \cup \{v\}) = \sum_{q \in Q} f_q \ * cost(q)$;
15. $g(v) = \frac{\mathrm{Cost}(Q,S') - \mathrm{Cost}(Q,S' \cup \{v\})}{\Delta T_v}$;
16. **if** $g(v) > gain\_benefit$ **then**
17. $gain\_benefit = g(v)$;
18. $v_0 = v$;
endif;
endfor;
19. **if** $(T' - \Delta T_{v_0}) > 0$ **then**
20. $T' = T' - \Delta T_{v_0}$;
21. $S' = S' \cup \{v_0\}$;
22. **for** each $q \in Q$ **do**
23. **if** $Length(q, v_0) = cost(q)$;
24. $p(q) = v_0$;
/* q changes its parent view to v_0 */
endif;
endfor;
endif;
endwhile

Let us analyze the time complexity of the integrated algorithm. The major step is to find the shortest tree rooted at a source which takes $O(m' + m + n)$ time [2] due to that G_{VQ} is a DAG with m view vertices and n query vertices, where m' ($\leqslant m^2 + nm$) is the number of directed edges in G_{VQ}. The total time for running the `while` loop in the integrated algorithm is $O((m + n)m')$, and the algorithm thus takes $O(|S'|(m + n)m') = O(m^4 + m^3n + m^2n^2)$ time in the worst case. An efficient implementation of the integrated algorithm takes less time, which is described as follows. The weights associated with the edges in G_{VQ} are not changed during the materialization of views, so, it is not necessary to compute the single-source shortest path tree for each vertex

each time within the `whole` loop. Instead, the single source shortest path tree rooted at each vertex can be computed only once, which takes $O(m(m^2 + mn))$ time for the m trees. Thus, the total time for the `while` loop is $O(|S'||V - S'||Q|) = O(m^2 n)$, and the whole algorithm requires $O(m^2 n + m^3)$ time.

5. Conclusions

The view selection problem under the maintenance time constraint is a fundamental problem in the design of data warehousing. In this paper, several heuristic algorithms for the problem have been proposed. The novelty of the proposed algorithms is finding good heuristic functions and reducing the problem to some well-established optimization problems. In turn, an approximate solution for the well known optimization problem will give a feasible solution for the problem. It must be mentioned that the proposed algorithms are significantly different from those greedy algorithms for solving the problem under the disk space constraint. In the presented case, the non-monotonicity of the problem under the maintenance time constraint makes it much harder to solve.

Acknowledgements

We would like to thank Jeffrey X. Yu for his helpful comments and suggestions about the proposed algorithms. The research by Weifa Liang is partially supported by a research grant from The Australian Research Council under a small grant schema (Grant No: F00025).

References

[1] E. Baralis, S. Paraboschi, E. Teniente. Materialized view selection in a multidimensional databases, in: Proceedings of the 23rd International Conference on VLDB, 1997, pp. 156–165.
[2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1994.
[3] A. Gupta, I. Mumick, Maintenance of materialized views: problems, techniques, and applications, IEEE Data Eng. Bull. 18 (2) (1995) 3–18.
[4] H. Gupta. Selection of views to materialize in a data warehouse, in: Proceedings of the Sixth ICDT, 1997, pp. 98–112.
[5] H. Gupta, V. Harinarayan, A. Rajaraman, J.D. Ullman. Index selection for OLAP, in: Proceedings of the International Conference on Data Engineering, Birmingham, UK, 1997, pp. 208–219.
[6] H. Gupta, I.S. Mumick. Selection of views to materialize under a maintenance cost constraint, in: Proceedings of the International Conference on Extended Database Theory, 1999, pp. 453–470.
[7] V. Harinarayan, A. Rajaraman, J.D. Ullman. Implementing data cubes efficiently, in: Proceedings of the 1996 ACM-SIGMOD Conference, 1996, pp. 205–216.
[8] Y. Kotidis, N. Roussopoulos. DynaMat: a dynamic view management system for data warehouses. Proc. of the 1999 ACM SIGMOD Conf., 1999.
[9] K.A. Ross, D. Srivastava, S. Sudarshan. Materialized view maintenance and integrity constraint checking: trading space for time, in: Proceedings of the 1996 ACM-SIGMOD Conference, 1996, pp. 447–458.
[10] D. Theodoratos, T. Sellis. Data warehouse configuration, in: Proceedings of the 23rd International Conference on VLDB, 1997, pp. 126–135.
[11] J. Yang, K.K. Karlapalem, Q. Li. Fast computation of sparse databases algorithms for materialized view design in data warehousing, in: Proceedings of the 23rd International Conference VLDB, 1997, pp. 136–145.
[12] A. Zelikovsky, A series of approximation algorithms for the acyclic directed Steiner tree problem, Algorithmica 18 (1997) 99–110.

Weifa Liang received his Ph.D. degree in computer science from The Australian National University in 1998. He received his M.E. degree in Computer Science from University of Science and Technology of China in 1989 and his B.S. degree in Computer Science from Wuhan University, China in 1984. He is currently a Lecturer in the Department of Computer Science at The Australian National University. His research interests include parallel processing, parallel and distributed algorithms, data warehousing and OLAP, query optimization, routing protocol design and graph theory.

Hui Wang is currently a Master (by research) student in computer science at School of Computer Science and Electrical Engineering in The University of Queensland. She received her B.S. degree in applied mathematics from Anhui University, China in 1984. Before coming to Australia, as a software engineer, she had been working in an institution in China for a decade to conduct research and development of application software in the simulation of VLSI circuits. Her current research interests include design and analysis of data warehousing, the consistency control of views in data warehousing, relational database applications, and Web development applications.

Maria E. Orlowska is currently the Professor in Information Systems at The University of Queensland in Australia. Since 1992 she has also acted as Distributed Unit Leader in the Cooperative Research Centre for Distributed Systems Technology (DSTC). She graduated with a Ph.D. (Computer Science) in June 1980 from the Institute of Applied Mathematics, Technical University of Warsaw. She is a trustee of the VLDB Endowment, and is a regular contributor to many other international conferences. She has published over 130 papers in international journals and conference proceedings. Her research expertise lies in the areas of: the theory of relational databases, distributed databases, various aspects of information systems design methodologies (including distributed systems), enhancement of semantic data modelling techniques by rigorous factors, transaction processing in distributed systems, concurrency control, distributed and federated database systems, and workflows technology.