Preview: Optimizing View Materialization Cost in Spatial Data Warehouses *

Songmei Yu, Vijayalakshmi Atluri, and Nabil Adam

MSIS Department and CIMIC
Rutgers University, NJ, USA
{songmei, atluri, adam}@cimic.rutgers.edu

Abstract. One of the major challenges facing a data warehouse is to improve the query response time while keeping the maintenance cost to a minimum. Recent solutions to tackle this problem suggest to selectively materialize certain views and compute the remaining views on-the-fly, so that the cost is optimized. Unfortunately, in case of a spatial data warehouse, both the view materialization cost and the on-the-fly computation cost are often extremely high. This is due to the fact that spatial data are larger in size and spatial operations are more complex and expensive than the traditional relational operations. In this paper, we propose a new notion, called preview, for which both the materialization and on-the-fly costs are significantly smaller than those of the traditional views. Essentially, to achieve these cost savings, a preview pre-processes the non-spatial part of the query, and maintains pointers to the spatial data. In addition, it exploits the hierarchical relationships among the different views by maintaining a universal composite lattice, and mapping each view onto it. We optimally decompose a spatial query into three components, the preview part, the materialized view part and the on-the-fly computation part, so that the total cost is minimized. We demonstrate the cost savings with realistic query scenarios.

1 Introduction

One of the major challenges facing a data warehouse is to improve the query response time while keeping the maintenance cost to a minimum. Recently, selectively materializing certain views over source relations has become the philosophy in designing a data warehouse. While materialized views incur the space cost and view maintenance cost, views that are not materialized incur on-the-fly computation cost. One has to balance both these costs in order to materialize the optimal views that incur minimum cost. This problem is exasperated when we consider a *spatial data warehouse* (SDW). This is because, spatial data are typically large in size (e.g., point, line, region, raster and vector images), and the operations on spatial data are more expensive (e.g., region merge, spatial overlay and spatial range selection). As a result, often, both on-the-fly computation cost and the view materialization cost are prohibitively expensive.

* The work is supported in part by the New Jersey Meadowlands Commission under the project Meadowlands Environmental Research Institute.

In this paper, we take a novel approach to resolve this issue. In particular, we introduce an intermediary view, called *preview*, for which both the materialization and on-the-fly costs are significantly smaller than those of the traditional views. Essentially, the idea of a preview is to pre-process the non-spatial part of the query and materialize this part based on certain cost conditions, but leave the spatial part for the on-the-fly and maintain pointers to the spatial data on which the spatial operation should be performed. In addition, a preview exploits the hierarchical relationships among different views. Obviously, storing previews in a data warehouse introduces overhead because it requires additional storage and process efforts to maintain the data sets during updates. However, we demonstrate that, the performance gain achieved through preview more than offsets this storage and maintenance overhead. Our ultimate goal is to optimize the total cost of a spatial data warehouse, which is the sum of the space cost of materialized views, the online computation cost of queries if not materialized, and the online computation and space cost of previews, if any.

This rest of the paper is organized as follows. We present the motivating example in Section 1. We present some preliminaries in Section 2. We introduce the Universal Composite Lattice in section 3. We define preview in Section 4. We discuss the related work in Section 5. We conclude our work in Section 6.

1.1 Motivating Example

In this section, we present an example that demonstrates that, for certain spatial queries, our approach to maintaining previews results in lower cost than optimally choosing a combination of on-the-fly and view materialization. Assume the spatial data warehouse comprising of a set of maps with their alphanumeric counterparts such as the area, the population amount and the temperature degree, as well as three basic metadata: location, time, and resolution. Assume that these maps specify different subjects of interest such as weather, precipitation, vegetation, population, soil, oil, or administrative region.

Now consider the following query that shows interests on a specific region: find the administrative boundary change of NJ area over last 10 years at 1m resolution level, and shows the vegetation patterns and population distributions within the same area, time frame and resolution level, and finally overlay the population maps and vegetation maps to deduce any relationships between them. The relation to store these data is called Map. An SQL-like query to specify this is as follows: select boundary(M.admin_map), M.vegetation_map, M.population_map, overlay(M.vegetation_map, M.populationa_map) from Map where Map.resolution = 1m AND Map.location = NJ AND 1994 < Map.year < 2005. For the purposes of execution, this query p can be visualized as having four parts, q_1, q_2, q_3 and q_4:

1. q_1: a spatial selection that retrieves boundaries of NJ administrative maps for last ten years on 1 m resolution.
2. q_2: a spatial selection that retrieves vegetation maps in NJ area for last ten years on 1m resolution.

3. q_3: a spatial selection that retrieves population maps in NJ area for last ten years on 1m resolution.
4. q_4 : a spatial join that overlays the results of q_2 and q_3 . Hence q_2 and q_3 are intermediate views for q_4.

The on-the-fly computation cost for each operation (q_1, q_2, q_3, q_4) is 4, 2, 2, 10 (s/image), and the space costs for admin_map boundary, vegetation_map and population_map are 5.0, 7.2, 6.0 (MB) respectively. Given a query q, we assume $S(q)$ denotes the space cost, $C(q)$ denotes the on-the-fly computation cost, and $T(q) = S(q) + C(q)$ denotes the total cost. For the sake of this example, we assume $S(q)$ is measured in Mega Bytes, and $C(q)$ in seconds. When computing $T(q)$, we assume 1MB translates into 1 cost unit and 1sec translates into 1 cost unit. Now let us consider the cost of the above query in the following four cases:

1. The entire query p is materialized. In other words, we materialize the result of q_1 and q_4. $T(p) = S(q_1)+S(q_4) = 5.0\times 10+(7.2+6.0)\times 10 = 50+132 = 182$.
2. The entire query p is computed on-the-fly. $T(p) = C(q_1) + C(q_2) + C(q_3) + C(q_4) = (4\times 10)+(2\times 10)+(2\times 10)+(10\times 10) = 40+20+20+100 = 180$.
3. Materialize q_1 and perform on-the-fly computation of q_4. Then $T(p) = S(q_1) + C(q_4) = 50 + (20 + 20 + 100) = 190$.
4. Materialize q_4 and perform on-the-fly computation of q_1. Then $T(p) = S(q_4) + C(q_1) = 132 + 40 = 172$.

Obviously one can choose the one among the alternatives that provides the highest cost savings. Now let us examine how using previews can reduce the view materialization cost. Let us assume we store the preview of q_1 and materialize q_4. Specifically, for q_1, we materialize the non-spatial part because its cost is below our pre-set threshold, therefore we store the metadata(NJ, 1995-2004, 1m) and pointers to the New Jersey administrative maps from year 1995 to 2004 and leave the spatial operation textitoverlay on-the-fly. Compared to materializing 10 years boundaries of administrative maps, the space and maintenance cost of storing preview is much cheaper than storing the spatial view itself. Compared to perform on-the-fly computation of retrieving 10 years boundaries of administrative maps, the query response time will be reduced by adding pointers. In another word, we reduce some on-the-fly computation cost of q_1 by paying price of storing its preview, so that the overall cost is optimized. For q_4, we still materialize it due to the very expensive *overlay* operation. The total cost of building a preview is the space cost of storing the preview the on-the-fly computation cost starting from the preview. In this real example, the space of using one row to store the preview is 0.01MB and the online boundary retrieval takes 2 second for each map. We use $PC(q)$ to denote the preview cost of query q, therefore:

1. $PC(q_1) = S(q_1) + C(q_1) = (0.01 \times 10) + (2 \times 10) = 0.1 + 20 = 20.1$
2. $S(q_4) = 132$
3. $T(p) = PC(q_1) + S(q_4) = 20.1 + 132 = 152.1$

Compared to the costs of previous methods, the total cost of query p is further optimized by constructing previews of q_1. In the next sections, we will present the definition of preview, and how we select appropriate set of queries for preview to optimize the total cost of an SDW.

2 Spatial Queries

In this section, we briefly present several important concepts. First, we define the basic algebra expression that is needed for constructing a *spatial query*. We then define an *atomic spatial query*, which serves as the smallest cost unit by decomposing a spatial query. We finally introduce a process denoted as *spatial projection*, which will be used to generate a preview.

The *hybrid algebra*, including hybrid relations R, hybrid operators *op* and hybrid operands X, constitutes the basis for defining a spatial query in a spatial data warehouse. Within an SDW, a base relation is a *hybrid relation* that includes attributes and tuples from both alphanumeric relations and spatial relations. For spatial relations, we adopt the definitions from the standard specifications of Open Geospatial Consortium (OGC). The spatial data types supported by this standard are from Geometry Object Model (GOM), where the geometry class serves as the base class with sub-classes for *Point, Curve (line)* and *Surface (Polygon)*, as well as a parallel class of *geometry collection* designed to handle geometries of a collection of points, lines and polygons. Conceptually, spatial entities are stored as relations with geometry valued attributes as columns, and their instances as rows. The hybrid operators *op* combine a complete set of relational operators *rop* $(\sigma, \pi, \cup, -, \times)$, comparison operators *cop* $(=, <, \leq, \geq, >, \neq)$, aggregate operators *aop* (distributive functions, algebraic functions, holistic functions) and spatial operators *sop* defined by OGC (Spatial Basic Operators, Spatial Topological Operators, Spatial Analysis Operators), or $op \in (rop \cup cop \cup aop \cup sop)$. A *hybrid algebra operand* is a distinct attribute of a hybrid relation, which could be either spatial operand or non-spatial operand. Now we define a spatial query based on the hybrid algebra.

Definition 1 *Spatial Query. A spatial query is a hybrid algebra expression* F*, which is defined as: (i) a single formula* f*, can be either unary* $(op(X_1))$*, binary* $(op(X_1, X_2))$*, or n-nary* $(op(X_1, \ldots, X_n))$*, where op is a hybrid algebra operator and each* X_i *is a hybrid operand, (ii) if* F_1 *is a hybrid algebra expression, then* $F = op(X_1, \ldots, X_m, F_1)$ *is a hybrid algebra expression, and (iii) if* F_1 *and* F_2 *are two hybrid algebra expressions, then* $F_1 \wedge F_2$*,* $F_1 \vee F_2$*,* $\neg F_1$ *and* (F_1) *are hybrid algebra expressions.*

In our motivating example, the spatial query is to retrieve the boundaries of administrative maps and overlaid results of vegetation maps and population maps under certain conditions from the hybrid relation Map. Each spatial query is composed of one or more atomic spatial queries, or $p = \{q_1, \ldots, q_n\}$, which is defined as follows:

Definition 2 *Atomic Spatial Query. Given a spatial query p, an atomic spatial query q is a component query within p, which is a hybrid algebra expression aF such that it contains only a single spatial operator sop.*

An atomic spatial query essentially is nothing but an atomic formula that serves as the smallest unit for the spatial operation cost measurement purpose. In addition, an atomic spatial query q can be composed of two parts, the spatial part and the non-spatial part. The spatial part includes a single well-defined spatial operator, and the non-spatial part could include the traditional selection-projection-join operations, comparison operations and aggregate operations. For each q, if we want to construct a preview for it, we need to perform spatial projection defined as follows:

Definition 3 *Spatial Projection. Let q be an atomic spatial query. The spatial projection of q, denoted as q^s, has only spatial operators.*

Essentially, a spatial projection of an atomic spatial query is computed by simply removing all non-spatial operations as well as all the operands associated with these operators. It comprises of only one spatial operation since by definition, the preview contains one spatial operation to begin with.

3 The Universal Composite Lattice

In this section, we define the Universal Composite Lattice (UCL), which captures the hierarchical relationships among all the possible queries in a given spatial data warehouse. UCL is essentially constructed by composing all its dimension hierarchies together. We first introduce a single dimension hierarchy.

3.1 The Single Dimension Hierarchy

For any given data warehouse, each attribute or dimension may vary from more general to more specific; the relationships thus mapped are called the dimension hierarchies or attribute concept hierarchies. Now we formally define the single dimension hierarchy, following the lines in [1].

Definition 4 *Single Dimension Hierarchy. Given an attribute d, we say there exists an edge from node h_i to node h_j, $h_i \rightarrow h_j$, in the dimension hierarchy H of d, if h_i is a more general concept than h_j, denoted as $h_i > h_j$.*

Here h_i and h_j are two nodes in the dimension hierarchy of d. Generally an attribute could have as many nodes as the user specified to capture the relationships among the different levels of the generalization of the dimension. The resultant dimension hierarchy may be a partial order. In figure 1, we present this single dimension hierarchy for each metadata. The concept hierarchy provides a basic framework for the query dependency relationship. Given two nodes h_i and h_j in H, we say there exists a dependency relationship between h_i and h_j if there exists $h_i \rightarrow h_j$. The dependency relationship indicates that the query represented at node h_i can be built by that represented at h_j. In other words, if one materializes the view at h_j, the query at h_i can be answered by simply generalizing the view at h_j. For example, we could generate a map of a country by combining maps of each state in that country, hence we say the query on the country depends on the query on the states. In this way, we can use the lower level query result to answer higher level queries instead of computing from scratch, which has been demonstrated to be an efficient query optimization technique [1].

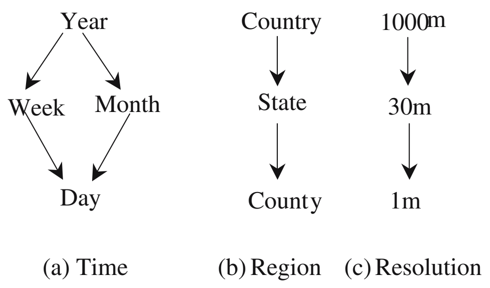

Fig. 1. The single dimension hierarchy

3.2 The Universal Composite Lattice

The Universal Composite Lattice (UCL) is built by integrating all the dimension concept hierarchies from a set of attribute domains $D = \{d_1, \ldots, d_k\}$. Therefore, we can use the UCL to represent the hierarchical relationships for all the queries in this data warehouse, and any input query can be mapped into this composite lattice and be evaluated based on its sub-queries. Suppose N_i be the set of nodes in the dimension hierarchy of d_i. Assuming a spatial data warehouse comprises of dimensions $D = \{d_1, \ldots, d_k\}$ of the spatial measures, then the UCL could at most have $(N_1 \times \ldots \times N_k)$ nodes. We define a universal composite lattice as follows.

Definition 5 *Universal Composite Lattice. Let $D = \{d_1, \ldots, d_k\}$ be the set of dimensions in SDW. Each node u in UCL is of the form $u = (n_1, \ldots, n_k)$ such that $n_1 \in N_1$ or null , $n_2 \in N_2$ or null, ..., $n_k \in N_k$ or null. There exists an edge $u_i \rightarrow u_j$, iff every $n_{ik} > n_{jk}$.*

Essentially, a universal composite lattice (UCL) is a directed graph that describes the query dependency relationships for a given spatial data warehouse. Every node in UCL is comprised of at least one node from the each dimension hierarchy or a null. The edge in UCL, as in the single dimension hierarchy represents that the higher level view represented by that node can be constructed from lower level views. Figure 2 shows the UCL constructed by combining the three dimension hierarchies of in figure 1. For the sake of simplicity, we have used the total order for the Time dimension. Generally, for any data warehouse, one can construct such a lattice to indicate the dependency relationships among different queries. The big advantage of this lattice is that every atomic spatial query can be mapped to some node on UCL. We call such mapping process *UCL instantiation*. We will introduce our notion of previews and how UCL instantiation help us to exploit the existing views when computing certain queries on-the-fly.

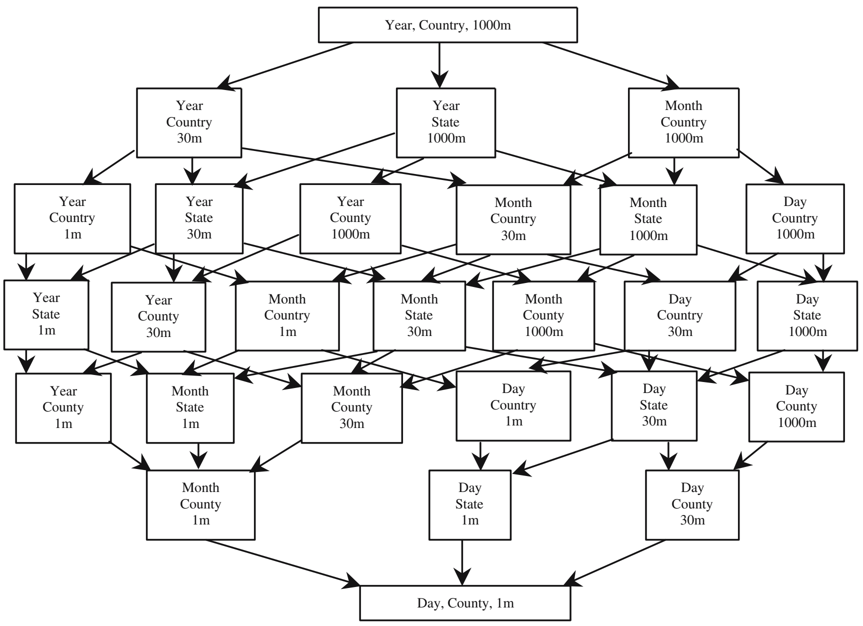

Fig. 2. A sample universal composite lattice

4 The Preview

Essentially, the preview of an atomic query comprises of the view of the preprocessed non-spatial part of the query, and the information necessary to compute the spatial part on-the-fly. As such it maintains pointers to the spatial objects on which the spatial operation should be performed. In addition, a preview also exploits the dependency relationships among different previews. A preview is formally defined as follows:

Definition 6 *Preview. Let q be an atomic spatial query. The preview of q, denoted as $pre(q)$, is a 4-tuple $\langle M, sop, O, V \rangle$, where: (1) M is non-spatial parts of q, (2) sop is the spatial operator, (3) O is a set of pointers to spatial objects, and (4) V is a set of pointers to all the sub-views that q depends on.*

By constructing a preview, we need to first do spatial projection of an atomic spatial query by separating spatial and non-spatial parts. Then we decide if we need to materialize non-spatial operations of q based on some pre-set cost threshold r to get M. Or, if the cost is greater than r, we materialize it otherwise we leave it on the fly. We also keep O, the set of pointers to the spatial objects. Then we extract spatial operator sop, which will be computed on the fly when q is executed. Finally we construct the pointer set V which points to all views or previews at the lower dependent level by instantiating the given UCL.

For example, in the motivating example, we perform the traditional selection and projection on q_1 and store ⟨((1995-2004), New Jersey, 1m), (boundary), (ptr1-ptr10)$\rangle$ as its preview $pre(q_1)$, where the non-spatial part $M = \langle$1995-2004, New Jersey, 1m $\rangle$ is materialized by SPJ operations, the pointers $(ptr_1 - ptr_{10})$ to maps are projected out based on certain conditions. Since there is one spatial operator involved, we put *boundary* in the operator position for the on-the-fly computation. V could include one or more pointers depending on how many sub-views are available. This preview is stored as a tuple in the data warehouse for further query evaluations.

Now we show how a preview can be mapped onto a UCL, and the pointer set pointing to the sub-views can be constructed accordingly. Generally, for any atomic spatial query, it will be either materialized, computed on-the-fly or built for a preview. UCL instantiation includes not only mapping the previews but also mapping the materialized views or the views computed on-the-fly. For simplicity, we only show mapping a preview onto UCL, and other mappings of a materialized view or a view computed on-the-fly can be conducted similarly with straightforward extensions.

As we introduced before, an SDW comprises of a dimension set $(d_1, \ldots, d_k)$ with dimension hierarchies set $(N_1, \ldots, N_k)$ for each dimension. Each specific node $u = (n_1, \ldots, n_k)(n_i \in N_i, i = 1, \ldots, k)$ has corresponding actual values stored in the base tables of the data warehouse, which is denoted as VL_i. For example, Year is one hierarchy in dimension Time, and its corresponding actual value in the data warehouse is a complete set or subset of (1980-2005). Generally, we denote $u = VL_i(i = 1, \ldots, k)$ iff VL_i is the set of actual values associated to u. For example, in the figure 2, $\langle$Year, State, resolution $\rangle = \langle$(1995-2004), New Jersey, 1m $\rangle$. Given a UCL, a simple linear search algorithm can map a preview of an atomic spatial query q, denoted as $pre(q)$, onto a given UCL (omitted due to space limit).

This algorithm basically performs linear search from the lowest level node to the highest level node in the UCL, and see if the M of a $pre(q)$ includes the actual hierarchy values of certain node. If we find this match, we add a pointer from that node to the $pre(q)$. Therefore we map a preview to an actual node in the UCL. In addition to the previews, we assume the materialized views and views that computed on-the-fly are also mapped onto the UCL, which instantiate the UCL for a spatial data warehouse. Hence if there are any materialized views or previews mapped there, we add a pointer from $pre(q_1)$ to those lower level views, or sub-views. Basically, $V = (t_1, \ldots, t_n)$ where $(t_i, i \in (1, n))$ is a pointer to one sub-view of $pre(q_1)$.

5 Related Work

A lot of work has been done in the area of optimizing cost of a data warehouse. Most of their work deal with selective materialization to reduce the total cost. In the initial research done on the view selection problem, Harinarayan et al. in [2] present algorithms for the view-selection problem in data cubes under a disk-space constraint. Gupta et al. extend their work to include indexes in [3]. Stefanovic et al. in [4] introduce the spatial data warehouse concept and object based selective materialization techniques for construction of spatial data cubes.

Karlo et al. [5] show that the variation of the view-selection problem where the goal is to optimize the query cost is inapproximable for general partial orders. Furthermore, Chirkova et al. in [6,7] show that the number of views involved in an optimal solution for the view-selection problem may be exponential in the size of the database schema, when the query optimizer has good estimates of the sizes of the views. Besides the theoretical research, there has been a substantial amount of effort on developing heuristics for the view-selection problem that may work well in practice. Kalnis et al. in [8] show that randomized search methods provide near-optimal solutions and can easily be adapted to various versions of the problem, including existence of size and time constraints. Recently, certain works have been done on how to materialize views for some specific systems or to answer queries more efficiently. Specifically, Karenos et al in [9] propose view materialization techniques to deal with mobile computing services, Liu et al in [10] compare two view materialization approaches for medical data to improve query efficiency, Theodoratos et al [11,12,13] build a search space for view selections to deal with evolving data warehousing systems, and Wu et al in [14] work on Web data to rewrite queries using materialized views.

However, all of their methods fall into two categories, i.e. either materialize a view or compute it on the fly. Our work presented in this paper differs from the above works in that given the specialty of spatial operations involved in a query, we design a third technique, *preview*, between view materialization and on-the-fly computation, which delivers a provably good solution with cost minimization for a spatial query and eventually a whole spatial data warehouse.

6 Conclusions

A spatial data warehouse integrates alphanumeric data and spatial data from multiple distributed information sources. Compared to traditional cases, a spatial data warehouse has a distinguished feature in that both the view materialization cost and the on-the-fly cost are extremely high, which is due to the fact that spatial data are larger in size and spatial operations are more expensive to process. Therefore, the traditional way of selectively materializing certain views while computing others on the fly does not solve the problem of spatial views.

In this paper we have dealt with the issue of minimizing the total cost of a spatial data warehouse while at the same time improve the query response time by considering their inter-dependent relationships. We first use a motivation example in realistic query scenarios to demonstrate the cost savings of building a preview. We then formally define preview, for which both the materialization and on-the-fly costs are significantly reduced. Specifically, a preview pre-processes the non-spatial part of the query, leaves the spatial operation on the fly, and maintains pointers to the spatial data. In addition, we show that a preview exploits the hierarchical relationships among the different views by maintaining a Universal Composite Lattice built on dimension hierarchies, and mapping each view onto it. We optimally decompose a spatial query into three components, the preview part, the materialized view part and the on-the-fly part, so that the total cost is minimized.

References

1. Han, J., Kamber, M. In: Data Mining: Concepts and Techniques. 1 edn. Morgan Kaufman Publishers (2001)
2. Harinarayan, V., Rajaraman, A., Ullman, J.: Implementing data cubes efficiently. In: Proc. of SIGMOD. Lecture Notes in Computer Science, Springer (1996) 205–216
3. Gupta, H., Mumick, I.: Selection of views to materialize in a data warehouse. Transactions of Knowledge and Data Engineering (TKDE) **17** (2005) 24–43
4. Stefanovic, N., Jan, J., Koperski, K.: Object-based selective materialization for efficient implementation of spatial data cubes. IEEE Transactions on Knowledge and Data Engineering(TKDE) **12** (2000) 938–958
5. Karlo, H., Mihail, M.: On the complexity of the view-selection problem. In: Proc. of Principles of Databases. Lecture Notes in Computer Science, Springer (1999)
6. Chirkova, R.: The view selection problem has an exponential bound for conjunctive queries and views. In: Proc. of ACM Symposium on Principles of Database Systems. (2002)
7. Chirkova, R., Halevy, A., Suciu, D.: A formal perspective on the view selection problem. In: Proc. of Internaltional Conference on Very Large Database Systems. (2001)
8. Kalnis, P., Mamoulis, N., Papadias, D.: View selection using randomized search. Data and Knowledge Engineering(DKE) **42** (2002)
9. Karenos, K., Samaras, G., Chrysanthis, P., Pitoura, E.: Mobile agent-based services for view materialization. ACM SIGMOBILE Mobile Computing and Communications Review **8** (2004)
10. Liu, Z., Chrysanthis, P., Tsui, F.: A comparison of two view materialization approaches for disease surveillance system. In: Proc. of SAC. Lecture Notes in Computer Science, Springer (2004)
11. Theodoratos, D., Ligoudistianos, S., Sellis, T.: View selection for designing the global data warehouse. Data and Knowledge Engineering (DKE) **39** (2001)
12. Theodoratos, D., Sellis, T.: Dynamic data warehouse design. In: Proc. of Data Warehousing and Knowledge Discovery. Lecture Notes in Computer Science, Springer (1999)
13. Theodoratos, D., Xu, W.: Constructing search spaces for materialized view selection. In: Proc. of the 7th ACM International Workshop on Data Warehousing and OLAP. Lecture Notes in Computer Science, Springer (2004)
14. Wu, W., Ozsoyoglu, Z.: Rewriting xpath queries using materialized views. In: Proc. of the Intl. Conference on Very Large Database Systems. Lecture Notes in Computer Science, Springer (2005)