Dynamic View Selection for OLAP

Michael Lawrence and Andrew Rau-Chaplin

Faculty of Computer Science
Dalhousie University
Halifax, NS, Canada
B3H 1W5
{michaell, arc}@cs.dal.ca
www.cgmLab.org

Abstract. Due to the increasing size of data warehouses it is often infeasible to materialize all possible aggregate views for Online Analytical Processing. View selection, the task of selecting a subset of views to materialize based on knowledge of the incoming queries and updates, is an important and challenging problem. In this paper we explore *Dynamic View Selection* in which the distribution of queries changes over time, and a subset of a materialized view set is updated to better serve the incoming queries.

1 Introduction

In a data warehousing environment, users interactively pose queries whose answers are used to support data-driven decision making. Such queries usually make heavy use of aggregation, which may be realized using the `GROUP-BY` clause in SQL. Since aggregate queries are so common and their results are typically very expensive to compute, aggregate views of the data are often pre-computed and stored in OLAP systems in order to speed up future query processing. From the perspective of efficient query answering, ideally all views would be pre-computed and made available for answering aggregate queries, however realistically storage and computational constraints limit the number of views that are usefully pre-materialized.

The problem of choosing a set of views for materialization is known as the *View Selection Problem*. In the view selection problem one wishes to select a set of views for materialization which minimizes one or more objectives, possibly subject to one or more constraints. Many variants of the view selection problem have been studied including minimizing the query cost of a materialized view set subject to a storage size constraint [1,2,3,4], minimizing a linear combination of query cost and maintenance cost of a materialized view set [5,6,7,8,9], and minimizing query cost under a maintenance cost constraint [3,10,11,12,13]. Note however that in most of these cases the problem considered is *static* in that: 1) Query frequencies are assumed to be static (i.e., not changing over time), and 2) It is assumed that the pool of materialized views is to be selected and computed from scratch rather than making use of a running OLAP system's pool of previously materialized views. While the static view selection problem is important, it captures only the start-up phase of a OLAP system and does not address what is arguably in practice the more important dynamic question: how, given a running OLAP system with an existing pool of materialized views and a new vector of query frequencies, should we identify views that should be added to our materialized pool and views that should be removed in order to best minimize query times subject to a storage size constraint?

The need for dynamic view management was forcefully made by Kotidis et al. in their DynaMat system [14]. As they observe, "This static selection of views [...] contradicts the dynamic nature of decision support analysis." There are a number of ways to approach the dynamic view selection problem, which we review in Section 2.2. In this paper we explore an alternative approach to *Dynamic View Selection*. We consider an OLAP system with two phases of operation: *Startup* and *Online*. In the Startup Phase an initial set of views must be selected based on some estimated query probabilities. This is the classical (static) view selection problem. In the Online Phase an "in use" OLAP system is considered, for which a set of views M has already been selected and materialized. Since over time the relative importance of each type of aggregate query may change due to the changing demands of its users, the system may elect to select a new set of views, M', which better serves the incoming queries. However, view materialization is computationally expensive and the time window in which new views can be materialized may prohibit selection of an entirely new view set. Thus the problem becomes selecting a new view set M' by discarding some views from M, and adding new ones to be materialized. We refer to the problem of incrementally updating a view set as the *Online View Selection Problem* (see Section 2). We believe that the online view selection problem is an important addition to the static variant, as OLAP systems in practice are restarted from scratch only infrequently and must be able to tune their performance to changing conditions on-the-fly.

Our approach to online view selection is to adapt methods that have proven to be effective for the static variant. In this paper we develop online adaptations of the greedy heuristic, BPUS, introduced by Harinarayan et al. [1] and three randomized techniques (iterative improvement, simulated annealing and two-phase optimization) initially proposed for static view selection by Kalnis et al. [3] (see Sections 3 and 4). Our challenge is two-fold. For the static phase, the randomized methods must be adapted so as to take into account maintenance cost in addition to the space constraint and for the online phase all of the methods must be adapted to take into account the existing pool of previously materialized views.

2 Problem Definition and Related Work

2.1 Static View Selection

A typical data warehouse stores its information according to a star schema having a central fact table with d feature attributes (dimensions), and some number of measure attributes. Queries to the data warehouse request aggregated measures from the perspective of some subset of the dimensions in the fact and dimension tables, which can be specified using the `GROUP-BY` clause in SQL. The aggregated table from which a query's results are collected is called a *view*, and is identified by the dimensions selected. Harinarayan et al. introduced the data cube lattice in [1] expressing the relationship between views as a partial order. There is a path from a view v_1 to a view v_2 in the lattice if queries on v_2 can be answered also using v_1 (although likely at a higher cost). The total number of views in the lattice is exponential in the number of dimensions.

The view selection problem can be formally defined as follows. For each view v in the lattice L, we have some estimate of the number of records r_v in v, and the frequency of queries f_v on v. As in most previous studies, we adopt the linear cost model presented in [1], where the cost of answering a query on a view v is r_v. The cost $q(v, M)$ of answering aggregate queries on view v using a materialized view set M is equal to the number of records in the smallest view in M which is an ancestor of v in the data cube lattice. The overall query time using M is a weighted sum of these terms

$$Q(M) = \sum_{v} f_v q(v, M),$$

and the size $S(M)$ of M is simply the sum of the sizes of each of the views in M. The update or maintenance cost $u(v, M)$ of a materialized view v in M is modeled based on a maintenance cost which is assigned to every edge (v_1, v_2) in the lattice. This represents the cost of maintaining v_2 using updates from v_1, and the maintenance cost $u(v, M)$ is the smallest maintenance cost over all paths from materialized ancestors of v_1 to v_2. Each node v also has an update frequency g_v, and the total update cost for a set of materialized views M is

$$U(M) = \sum_{v \in M} g_v u(v, M).$$

Note that while $Q(M)$ decreases when new views are added, $U(M)$ does not always increase, as the additional cost of maintaining a materialized view v might be outweighed by the benefit that v has in propagating smaller batches of updates to its children. Our goal for the static phase is as follows: to select a M which has the minimum $Q(M) + U(M)$ subject to the constraint $S(M) < S_{max}$ for some maximum size S_{max}.

Numerous solutions have been proposed to the (static) view selection problem on data cubes. The first is a greedy algorithm presented by Harinarayan et al. [1], which is proven to find a solution within 63% of optimal. In [5] the same heuristic was extended to minimize sum of query and update cost. Shukla et al. give a heuristic minimizing query cost in [2] which is asymptotically faster than that of [1], but achieves the same solution only under certain conditions. In [6] a greedy algorithm for minimizing the sum of query and update cost is given, their update cost modeling is more accurate, however when choosing a view v to select they only consider the update cost of v and not its impact on the entire view set. Gupta gives the first solution to the view selection problem minimizing query cost under an update cost constraint in [13], followed by two algorithms of Liang et al. in [10], and genetic algorithm approaches in [12,11] In [15] Agrawal et al. present a tool and algorithms for selecting a set of views based on a cost metric involving query cost, update cost, index construction and other factors. Nadeau and Teorey give a greedy algorithm minimizing query cost under a space constraint which is polynomial in the number of dimensions [4], but does not perform as well as other greedy heuristics. Kalnis et al. use randomized algorithms to search the solution space of view sets in [3], minimizing query cost and constrained by space or update cost.

2.2 Online View Selection

For the online phase we consider a data warehousing system which has a fixed-sized time window to materialize new views which are not currently materialized, but are perhaps more beneficial to the changing query patterns of the users. However because of the space constraint, a number of views may have to be discarded as well. Based on the size of the available time window for computing new views, the database administrator calculates how much of the materialized view set can be replaced. We do not consider update costs in online view selection because updates themselves are counter to the purpose of online view selection. Online view selection is an act which is typically performed at regular maintenance intervals, based on an observed or expected change in query probabilities. During these intervals, updates are applied to the views and so we do not expect updates to be applied between intervals, hence update cost is not our concern in online view selection. Hence we can define the online view selection problem as follows: Given a materialized view set M and new query frequencies $\mathbf{f}'$, find a M' which minimizes $Q(M')$ with respect to the new query frequencies, and such that

$$\sum_{v \in M \cap M'} r_v \geq (1-h) \cdot S_{max},$$

for some h which represents a percentage of M (in terms of size) for which the system has enough resources to materialize new views for. Our dynamic view selection involves an initial startup phase consisting of a static selection of views M_1, and multiple online phases where M_1 is updated to M_2, M_2 to M_3 and so on. The decision of when to select an M_{i+1} by updating M_i can be made in many ways, for example during a pre-allocated maintenance window, when average query time degrades past an unacceptable level, or based on measuring the difference between the current query distribution and the distribution at the last online phase.

Our formulation of dynamic view selection is different from that of other studies [9,8,14]. In [9,8], Theodoratos et al. consider what they call dynamic or incremental data warehouse design. In the static phase, they are given a fixed set of queries Q, and views must be selected from multiquery AND/OR-DAGs which answer the queries with minimum cost. In the online phase, additional queries are added to the set Q, and new materialized views are added so that the new queries can be answered. In practice there may not be extra space available for materializing new views, and some previously materialized views must be discarded based on the fact that some queries are no longer of interest. In the dynamic view selection considered here, our materialized view set M is able to answer any possible aggregate query, and its size never increases beyond S_{max} over time. It may be the case that additional space is available for materialized views over time, which can easily be handled by our implementation of the algorithms.

In [14], Kotidis and Roussopoulos approach dynamic view selection by caching fragments of views. A view fragment is a portion of a whole view which results from a range selection on its dimensions. Their approach is fundamentally different from ours in that it can only choose to store aggregate data which has been requested from the user, where as pre-materializing a set of views is more flexible in that any aggregate data which can be produced is considered for storage. Also, their approach is in reaction to user's queries, where as a materialized view set approach aims to prepare the system for future queries. We believe ours to be a more valuable approach to dynamic view selection for the following reasons:

1. As argued in [16], the ad-hoc nature of OLAP queries reduces the chance that stored fragments will be able to fully answer future queries. Storing whole views guarantees that any queries on the same or more highly aggregated views can be answered.
2. Knowledge giving an expectation of future query loads (e.g. daily reports) may be available, allowing advance preparations to be made by choosing an appropriate pre-materialized view set.
3. Multiple unrelated and popular aggregate queries may have a common ancestor which can answer all of them at a slightly higher cost. In DynaMat, this ancestor will never be considered for storage unless it is queried, where as a good approach to view selection will store this ancestor instead of the individual aggregates below it, resulting in significant space savings which can be put to better use.

3 Randomized Algorithms for Dynamic View Selection

In order to apply randomized search to a problem, transitions between feasible solutions are required. Each search process moves stochastically through the graph of feasible solutions called the search space, which can be pictured as a topographical space where elevation represents objective value and locality represents connectivity of the solutions through the transitions defined. Since we are minimizing, the "lower" solutions in this space are the ones we desire. The effectiveness of a randomized search strategy depends on the shape of the search space and in what manner the search moves through it.

For the static phase we define two transitions, based on those in [3]:

1. Add a random view which is not in the current solution and randomly remove selected views as necessary to satisfy the space constraint.
2. Remove a randomly selected view.

The second transition is different from that of [3], which, after removing a view fills the rest of the available space with views. This is because our algorithms minimize both query and maintenance cost, as opposed to just maintenance cost. Under these conditions it is no longer safe to assume that the optimal view set is a "full" one, and the randomized algorithms must adapt to the tradeoff between query and maintenance cost. We similarly modify Kalnis et al.'s method of generating a random solution, by repeatedly adding views to an initially empty view set until the addition of some view v causes a decrease in overall cost, and removing v. This method gave us better results in terms of generating solutions nearer to the favourable ones than another technique of generating random solutions, which was to randomly pick a size S_0 from a uniform distribution on $[0, S_{max}]$, and creating a view set no larger than S_0 by adding as many random views as possible.

The three randomized search algorithms considered here are iterative improvement (II), simulated annealing (SA) and two-phase optimization (2PO), as described in [3].

- II makes a number of transitions, only accepting ones which lead to a better solution. When a number of unsuccessful transitions from a state are made (local minimum), II starts again from a random initial state. The search terminates after some maximum time or number of local minima.
- SA is an analogy to the process of cooling a physical system It works like II, except that uphill transitions may be accepted with some probability that is proportional to a "temperature" which decreases with time. The algorithm halts when the temperature reaches a fixed "freezing point", returning the best solution found during the process.
- 2PO combines II with SA. II is first applied to find a good local minimum, from which SA is applied with a small initial temperature to do a more thorough search of the surrounding area.

II tends to work well if the problem has structure so that good solutions are near each other in the search space. SA is more robust than II in that it can overcome the problem that good solutions may be near each other in search space, but separated by a small number of relatively worse solutions. 2PO attempts to combine the best of both II, which proceeds in a more direct manner towards a solution, and SA, which is able to more thoroughly explore an area.

We modify the transitions for online view selection as follows: If, while removing views to satisfy the space constraint, transition 1 violates the online constraint, then we add the previously removed view back to the solution, and continue only removing views in $M' - M$ from M' until the space constraint is satisfied. If transition 2 violates the online constraint, the removed view is re-added and a randomly selected view in $M' - M$ is removed instead.

4 Greedy Algorithm for Dynamic View Selection

For the static phase, the BPUS heuristic [1] begins with an initially empty view set, and greedily adds views which maximize a benefit heuristic. If the currently selected view set is M, then the benefit per unit space of adding an unselected view v is defined as

$$\frac{(Q(M)+U(M))-(Q(M\cup\{v\})+U(M\cup\{v\}))}{r_v},$$

i.e., the reduction in overall cost achieved by adding v, scaled by size. The BPUS algorithm adds the view v with maximum benefit per unit space until either there is no more space for materialized views, or v has negative benefit (when $U(M\cup\{v\})-U(M)>Q(M)-Q(M\cup\{v\})$). To apply BPUS to online view selection, the same heuristic is "reversed", and the objective cost only considers query time of the materialized view set. When deciding which views to remove from M, we choose the view v which minimizes

$$\frac{Q(M-\{v\})-Q(M)}{r_v},$$

the increase in average overall cost scaled to the size of v. Once a set of views totalling no more than $h\cdot S_{max}$ in size has been removed from M, we apply the BPUS heuristic (without update cost) to greedily select which views to replace them with, arriving at our solution M'.

5 Experimental Results

In our evaluation we use a variety of synthetic data sets which we can control the properties of in terms of size, dimensionality, skew, etc. In particular we focus on two classes of data sets: 1) *uniform*, where the cardinality of all dimensions are equal, representing data cube lattices with highly uniform view sizes, and 2) *2pow*, where the cardinality of the i-th dimension is 2^i, representing data cube lattices with highly skewed view sizes. In all cases the number of rows in the data sets was set to 1 billion and S_{max} was set to be 10 times this number of rows. We use two different types of query distributions: 1) *uniform random*, where query probabilities are assigned from a uniform random distribution, and 2) *hot regions* [17], where 90% of the queries are distributed amongst a set of views (the "hot region") selected from the bottom 1/3 of the lattice and containing 10% of the total views. The remaining 10% of the queries are distributed uniformly amongst the other views. We believe this to be a realistic but very challenging scenario, due to the underlying semantics of the dimensions of the data warehouse in which some combinations of dimensions may not provide useful information, while others do, hence there may be a large number of views which are simply not interesting at all.

We apply the randomized algorithms to view selection similarly as in [3], although our parameter selection is slightly different due to the different nature of the problem which now involves update costs. The selected parameters are shown in Table 1. Readers are referred to [3] for a description of the meaning of these parameters.

Table 1. Selected parameters for the randomized algorithms applied to both the static and online phases of dynamic view selection

	Static	Online
$cycles_{II}$	$50 \cdot 2^{d-10}$	$40 \cdot h \cdot 2^{3(d-10)/4}$
$min.d_{II}$	$7d$	$3d$
$cycles_{SA}$	$2^d/20$	$h/30\% \cdot 2^d/20$
T_{SA}	10^7	10^3
Δt_{SA}	0.9	0.8
$cycles_{2PO}$	$cycles_{II}/4$	$cycles_{II}/2$
$min.d_{2PO}$	$min.d_{II}$	$2d$
T_{2PO}	10^5	10^2
Δt_{2PO}	0.9	0.64

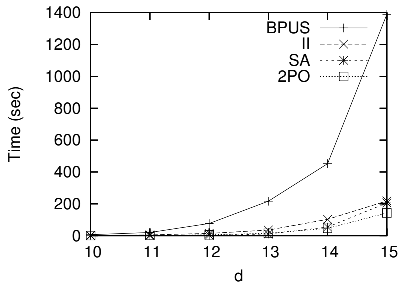
5.1 The Startup Phase: Static View Selection

In static view selection we are primarily concerned with 1) Quality: the solution values achieved by the randomized algorithms vs. that of BPUS, and 2) Efficiency: the amount of time it takes to converge on a solution. To maintain consistency with the literature [3] and since randomized algorithms are being considered as an alternative to BPUS, we express their solution quality as a factor of the solution quality of BPUS, called the *scaled solution* value.

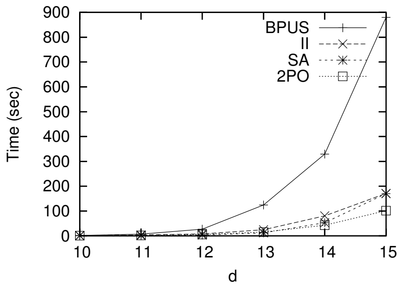
Figure 1 shows the running time of BPUS, and the time to convergence for all randomized algorithms as the number of dimensions is varied. As the plot shows, the randomized algorithms converge much faster than the BPUS heuristic for both uniform and highly skewed data, especially in larger dimensions.

Now we aim to establish scalability of the randomized techniques in terms of solution quality. Figure 2 shows the scaled cost of the solutions found by the randomized algorithms as the number of dimensions is increased. As the figure shows, the randomized algorithms perform competitively against the BPUS heuristic, with their solutions falling typically within a few percent of it and being especially close with uniform data. Surprisingly, there are classes of problem instances where some of the randomized algorithms outperform the heuristic by as much as 15%. These problem instances are ones for which the update cost is sufficiently prohibitive that the better solutions are ones which contain a small number of views. BPUS finds a maximal solution with respect to size, in that no more views can be added without increasing the overall cost. As a result the randomized algorithms are able to find solutions with a much better update cost, at the expense of a slightly higher query cost.

Although the results from Figure 2 may suggest the randomized algorithms are more favourable for static view selection, we note that this performance is only observed for such problem instances where the query and update cost are relatively balanced. When query cost is the dominant factor, BPUS significantly

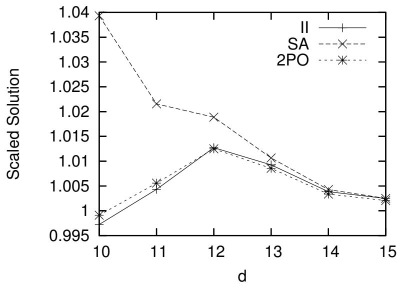


(a) Uniform Data/Uniform Queries

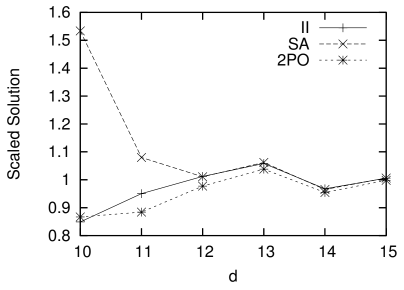


(b) Skewed Data/Hot Regions

Fig. 1. Running time vs dimensionality for static view selection with 10^9 rows and $S_{max} = 10^{10}$. The mean of 20 independent trials is shown.



(a) Uniform Data/Uniform Queries



(b) Skewed Data/Hot Regions

Fig. 2. Scaled solution vs d for static view selection with 10^9 rows and $S_{max} = 10^{10}$. The mean of 20 independent trials is shown.

outperforms the randomized algorithms. This is because, when update cost is not optimized as in [3], the randomized transitions can be designed knowing that the optimal view set will be a "full" one. However with the addition of update cost as an objective, the transitions must guide the randomized algorithms to the region of search space having view sets of the best size, as well as to a good choice of views in sets of that size. This adds another dimension of difficulty to the problem. The poor performance of the randomized algorithms when query cost is dominant is an indication that the randomized transitions have difficulty in guiding the search towards more full view sets. It is suggested that the randomized transitions of [3] be used if update cost is not expected to be prohibitive.

5.2 The Online Phase: Online View Selection

In the following tests each algorithm begins with the same initial view set M, which is selected using BPUS. $d = 12$ dimensions are used, with the parameters of the randomized algorithms summarized in Table 1. Unless otherwise indicated, $h = 30\%$ was chosen for the tests. For the uniform query distribution, the drift in query probabilities for a single iteration was achieved by scaling each view's probability by a random factor chosen uniformly between 0 and 1, and re-normalizing them so that they sum to 1. Using this query drift model the area between distribution curves on an iteration is generally in the range of 0.9 to 1.0 (with the maximum possible difference being 2). For the hot region query distribution, query drift was achieved by selecting both a beginning and ending hot region, and interpolating between the two over the iterations.

For performance on a single iteration, we are concerned with the improvement in query time. We measure the percent improvement of an online iteration from M to M' as

$$Imp(M, M') = 100\frac{Q(M) - Q(M')}{Q(M)}.$$

Note that the percent improvement not only depends on algorithm performance but also the amount that query distribution shifts.

First we examine the scalability of the algorithms in terms of running time. Figure 3 shows the results as we increase the dimensionality of the data sets. As in the case of static view selection, the randomized techniques are significantly faster than BPUS, especially with higher dimensions.

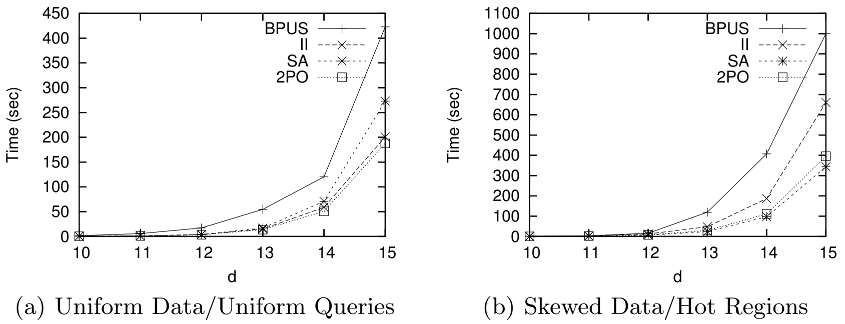
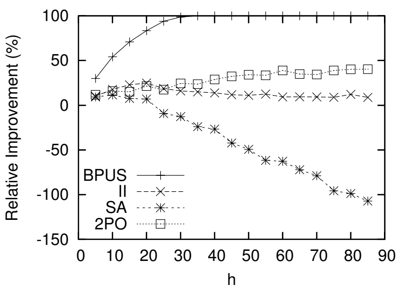
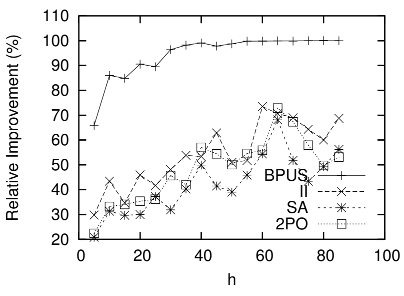


Fig. 3. Running time vs. dimensionality for online view selection with 10^9 rows, $S_{max} = 10^{10}$ and $h = 30\%$. The mean of 20 independent trials is shown.

One question which is pertinent to online view selection is how much can we improve the current materialized view set given that we only have time to replace $h\%$ of it, or how much of the view set must be replaced to achieve a given improvement in query time. Figure 4 shows the relative improvement in query time as h is increased. The relative improvement is the percent improvement $Imp(M, M')$ achieved relative to the percent improvement $Imp(M, M^{new})$ which can be achieved if an entirely new set of views M^{new} were selected with BPUS. 0% means that $Q(M') = Q(M)$ (no improvement), 100% means $Q(M') = Q(M^{new})$, and $< 0\%$ means that $Q(M') > Q(M)$ (negative improvement). From the figure we can see that the online version of BPUS is a very strong performer for both uniform and skewed data, able to make 95% of the improvement of a newly chosen view set by replacing as little as 30% of it. A



(a) Uniform Data/Uniform Queries



(b) Skewed Data/Hot Regions

Fig. 4. Percent improvement (relative to that of a newly selected view set) as h is increased. A 12-dimensional data cube is used. The mean of 20 independent trials is shown.

larger improvement with smaller h is possible for the hot regions instance, since replacing the views in the hot region is sufficient for considerable improvement.

6 Conclusions and Future Work

In this paper we have described a new approach to dynamic view selection which recognizes that in practice OLAP systems are restarted from scratch only infrequently and must be able to tune their performance to changing conditions on-the-fly. We have described a greedy and three randomized methods for dynamic view selection and implemented and evaluated them in the context of a large-scale OLAP system. Overall, in terms of solution quality, our BPUS-online adaptation appears to outperform the three randomized methods studied. However, as the number of dimensions grows the computational cost of BPUS-online may become impractically large and in this case the randomized methods presented here offer an attractive alternative. One important area of future work is to consider how best the dynamic view selection method proposed here can be combined with established caching and batch query optimization approaches.

References

1. V. Harinarayan, A. Rajaraman, and J. D. Ullman, “Implementing data cubes efficiently,” in *proc. SIGMOD ’96*, pp. 205–216, ACM, 1996.
2. A. Shukla, P. Deshpande, and J. F. Naughton, “Materialized view selection for multidimensional datasets,” in *proc. VLDB ’98*, pp. 488–499, Morgan Kaufmann, 1998.
3. P. Kalnis, N. Mamoulis, and D. Papadias, “View selection using randomized search,” *Data Knowl. Eng.*, vol. 42, no. 1, pp. 89–111, 2002.
4. T. P. Nadeau and T. J. Teorey, “Achieving scalability in OLAP materialized view selection,” in *proc. DOLAP ’02*, pp. 28–34, ACM, 2002.
5. H. Gupta and I. S. Mumick, “Selection of views to materialize in a data warehouse,” *Data Knowl. Eng.*, vol. 17, pp. 24–43, Jan. 2005.

6. H. Uchiyama, K. Runapongsa, and T. J. Teorey, "A progressive view materialization algorithm," in *proc. DOLAP '99*, pp. 36–41, ACM, 1999.
7. E. Baralis, S. Paraboschi, and E. Teniente, "Materialized views selection in a multidimensional database," in *proc. VLDB '97*, pp. 156–165, Morgan Kaufmann, 1997.
8. D. Theodoratos, T. Dalamagas, A. Simitsis, and M. Stavropoulos, "A randomized approach for the incremental design of an evolving data warehouse," in *proc. ER '01*, pp. 325–338, Springer, 2001.
9. D. Theodoratos and T. Sellis, "Incremental design of a data warehouse," *J. Intell. Inf. Syst.*, vol. 15, no. 1, pp. 7–27, 2000.
10. W. Liang, H. Wang, and M. E. Orlowska, "Materialized view selection under the maintenance time constraint," *Data Knowl. Eng.*, vol. 37, no. 2, pp. 203–216, 2001.
11. J. X. Yu, X. Yao, C.-H. Choi, and G. Gou, "Materialized view selection as constrained evolutionary optimization," in *IEEE Trans. on Syst., Man and Cybernetics, Part C*, vol. 33, pp. 458–467, IEEE, Nov. 2003.
12. M. Lee and J. Hammer, "Speeding up materialized view selection in data warehouses using a randomized algorithm," *J. Coop. Info. Syst.*, vol. 10, no. 3, pp. 327–353, 2001.
13. H. Gupta and I. S. Mumick, "Selection of views to materialize under a maintenance cost constraint," in *proc. ICDT '99*, pp. 453–470, Springer, 1999.
14. Y. Kotidis and N. Roussopoulos, "A case for dynamic view management," *J. Trans. Database Syst.*, vol. 26, no. 4, pp. 388–423, 2001.
15. S. Agrawal, S. Chaudhuri, and V. R. Narasayya, "Automated selection of materialized views and indexes in sql databases," in *proc. VLDB '00*, pp. 496–505, Morgan Kaufmann, 2000.
16. T. Loukopoulos, P. Kalnis, I. Ahmad, and D. Papadias, "Active caching of on-line-analytical-processing queries in www proxies," in *proc. ICPP '01*, pp. 419–426, IEEE, 2001.
17. P. Kalnis, W. S. Ng, B. C. Ooi, D. Papadias, and K.-L. Tan, "An adaptive peer-to-peer network for distributed caching of olap results," in *proc. SIGMOD '02*, pp. 25–36, ACM, 2002.