Materialized View Selection for Aggregate View Recommendation

Humaira Ehsan([✉]) and Mohamed A. Sharaf

School of Information Technology and Electrical Engineering,
The University of Queensland, Brisbane, QLD, Australia
{h.ehsan,m.sharaf}@uq.edu.au

Abstract. Data analysts arduously rely on data visualizations for drawing insights into huge and complex datasets. However, finding interesting visualizations by manually specifying various parameters such as type, attributes, granularity is a protracted process. Simplification of this process requires systems that can automatically recommend interesting visualizations. Such systems primarily work first by evaluating the utility of all possible visualizations and then recommending the top-k visualizations to the user. However, this process is achieved at the hands of high data processing cost. That cost is further aggravated by the presence of numerical dimensional attributes, as it requires binned aggregations. Therefore, there is a need of recommendation systems that can facilitate data exploration tasks with the increased efficiency, without compromising the quality of recommendations. The most expensive operation while computing the utility of the views is the time spent in executing the query related to the views. To reduce the cost of this particular operation, we propose a novel technique *mView*, which instead of answering each query related to a view from scratch, reuses results of the already executed queries. This is done by incremental materialization of a set of views in optimal order and answering the queries from the materialized views instead of the base table. The experimental evaluation shows that the *mView* technique can reduce the cost at least by 25–30% as compared to the previously proposed methods.

1 Introduction

With the unprecedented increase in the volume of data, the challenge of finding efficient ways to extract interesting insights is critical. As such, data visualization has become the most common and effective tool for exploring such insights. Generally, the visualizations are generated using user-driven tools like Tableau, Qlik, Microsoft Excel, etc. However, the use of these tools is of limited effectiveness for large datasets, as it is very difficult for the user to manually determine the best data visualization by sequentially browsing through the available representations. Research efforts are therefore being directed to propose recommendation systems that *automatically recommend visualizations* [3,4,8–10,13]. These systems automatically manipulate the user selected dataset, generate all possible visualizations, and recommend the top-k interesting visualizations, where interestingness is quantified according to some utility function such as deviation, similarity, diversification, etc. The generation of these all possible visualizations is challenged by a wide range of possible factors. This include user-driven factors such as individual user preferences, data of interest, information semantics and tangible factors such as chart type, possible attribute combinations and available transformations (e.g. sorting, grouping, aggregation and binning).

Recent studies have focused on automatically generating all possible *aggregate views* of data and proposing search strategies for finding the top-k views for recommendation, based on the deviation based utility metric [3,4,13]. The search space of all possible visualizations is huge and it explodes even further in the presence of *numerical dimensions*, as *binned aggregation* is required to group the numerical values along a dimension into adjacent intervals. In our previous work [3,4] on visualization recommendation, binning for numeric dimensions was introduced and efficient schemes (named as *MuVE*) to recommend the top-k binned views were proposed.

The most expensive operation while computing the utility of views is the time spent in executing the queries related to the views. To reduce the cost of this particular operation, a novel technique *mView* is proposed, which instead of answering each query related to a view from scratch, *reuses results* from the already executed queries. In summary, this is done by materializing views and answering queries from the materialized views instead of the base table. The idea of materializing views for reducing the query-processing time is well studied in the literature [2,6,11] and has proven significant relevance to a wide variety of domains, such as query optimization, data integration, mobile computing and data warehouse design [6,11]. However due to prohibitively large number of views, the blind application of materialization may result in even further degradation of the cost [2]. Substantial amount of work has already been done to select an appropriate set of views to materialize that minimize the total query response time and the cost of maintaining the selected views, given a limited amount of resource, e.g., materialization time, storage space etc. [5].

In this work our proposed technique *mView* first defines a cost benefit model to decide which views are the best to reuse. Later, in an optimal order it materializes the best set of views, which reduce the overall cost of the solution.

The main contributions of this work are as follows:

- We formulate and analyze the problem of selecting views to materialize for efficiently generating aggregate views in the presence of numerical attribute dimensions (Sect. 3).
- We propose the *mView* technique, which introduces a novel search algorithm, particularly optimized to leverage the specific features of the binned views (Sect. 4).
- We conduct extensive experimental evaluation, which illustrate the benefits achieved by *mView* (Sects. 5.1 and 5.2).

2 Preliminaries

2.1 Aggregate View Recommendation

The process of visual data exploration is typically initiated by an analyst specifying a query Q on a database D_B. The result of Q, denoted as D_Q, represents a subset of the database D_B to be visually analyzed. For instance, consider the following query Q:

$$\texttt{Q: SELECT} * \texttt{FROM } D_B \texttt{ WHERE T;}$$

In Q, T specifies a combination of predicates, which selects a portion of D_B for visual analysis. A visual representation of Q is basically the process of generating an aggregate view V of its result (i.e., D_Q), which is then plotted using some of the popular visualization methods (e.g., bar charts, scatter plots, etc.). Similar to traditional OLAP systems and recent data visualization platforms [8,9,12,13], our model is based on a multi-dimensional database D_B, consisting of a set of dimension attributes $\mathbb{A}$ and a set of measure attributes $\mathbb{M}$. Additionally, $\mathbb{F}$ is the set of possible aggregate functions over the measure attributes $\mathbb{M}$, such as `SUM`, `COUNT`, `AVG`, `STD`, `VAR`, `MIN` and `MAX`. Hence, an aggregate view V_i over D_Q is represented by a tuple (A, M, F) where $A \in \mathbb{A}$, $M \in \mathbb{M}$, and $F \in \mathbb{F}$. That is, D_Q is grouped by dimension attribute A and aggregated by function F on measure attribute M. A possible view V_i of the example query Q above would be expressed as:

$$V_i\texttt{: SELECT A, F(M) FROM } D_B \texttt{ WHERE T GROUP BY A;}$$

where the `GROUP BY` clause specifies the dimension A for aggregation, and $F(M)$ specifies both the aggregated measure M and the aggregate function F.

Typically, a data analyst is keen to find visualizations that reveal some interesting insights about the analyzed data D_Q. However, the complexity of this task stems from: (1) the large number of possible visualizations, and (2) the interestingness of a visualization is rather subjective. Towards automated visual data exploration, recent approaches have been proposed for recommending interesting visualizations based on some objective, well-defined quantitative metrics (e.g., [8,9,13]). Among those metrics, recent case studies have shown that a *deviation-based* metric is able to provide interesting visualizations that highlight some of the particular trends of the analyzed datasets [13].

In particular, the deviation-based metric measures the distance between $V_i(D_Q)$ and $V_i(D_B)$. That is, it measures the deviation between the aggregate view V_i generated from the subset data D_Q vs. that generated from the entire database D_B, where $V_i(D_Q)$ is denoted as *target* view, whereas $V_i(D_B)$ is denoted as *comparison* view. The premise underlying the deviation-based metric is that a view V_i that results in a higher deviation is expected to reveal some interesting insights that are very particular to the subset D_Q and distinguish it from the general patterns in D_B. To ensure that all views have the same scale, each target view $V_i(D_Q)$ is normalized into a *probability distribution* $P[V_i(D_Q)]$ and each comparison view into $P[V_i(D_B)]$.

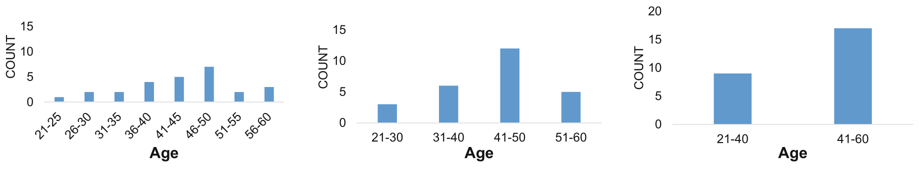
For a view V_i, given the probability distributions of its target and comparison views, the deviation $D(V_i)$ is defined as the distance between those probability distributions. Formally, for a given distance function *dist* (e.g., Euclidean distance, Earth Mover's distance, K-L divergence, etc.), $D(V_i)$ is defined as:

$$D(V_i) = dist(P[V_i(D_Q)], P[V_i(D_B)]) \tag{1}$$

Consequently, the deviation $D(V_i)$ of each possible view V_i is computed, and the k views with the highest deviation are recommended (i.e., *top-k*) [13]. Hence, the number of possible views to be constructed is $N = 2 \times |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$, which is clearly inefficient for a large multi-dimensional dataset.

2.2 Binned Views

In the previous section we discussed about aggregate view recommendation specifically for categorical dimensions. However, for continuous numerical dimensions, typically the numerical values along a dimension require grouping into adjacent intervals over the range of values. For example, consider a table of employees, which has *Age* as a numerical dimension attribute. Particularly, one of the aggregate views on this attribute is count the number of employees grouped by *Age*. For this type of views, it is more meaningful if adjacent intervals are grouped together and shown in a summarized way. For example, Fig. 1a shows the whole range grouped in 8 bins.



(a) $V_{i,8}$ (b) $V_{i,4}$ (c) $V_{i,2}$

Fig. 1. Generating $V_{i,2}$ by performing aggregation on $V_{i,4}$ or $V_{i,8}$

To enable the incorporation and recommendation of visualizations that are based on numerical dimensions, in our previous work [3,4], we introduced the notion of a *binned view*. A binned view $V_{i,b}$ simply extends the basic definition of a view to specify the applied binning aggregation. Specifically, given a view V_i represented by a tuple (A, M, F), where $A \in \mathbb{A}$, $M \in \mathbb{M}$, $F \in \mathbb{F}$, and A is a continuous numerical dimension with values in the range $L = [L_{min} - L_{max}]$, then a binned view $V_{i,b}$ is defined as:

Binned View: Given a view V_i and a bin width of w, a binned view $V_{i,b}$ is a representation of view V_i, in which the numerical dimension A is partitioned into a number of b equi-width non-overlapping bins, each of width w, where $0 < w \leq L$, and accordingly, $1 \leq b \leq \frac{L}{w}$.

For example, Fig. 1a shows a binned view $V_{i,8}$, in which the number of bins $b = 8$ and the bin width $w = 5$, while Fig. 1c shows a binned view $V_{i,2}$, in which the number of bins $b = 2$ and the bin width $w = 20$. Note that this definition of a binned view resembles that of an equi-width histogram in the sense that a bin size w is uniform across all bins. While other non-uniform histograms representations (e.g., equi-depth and V-optimal) often provide higher accuracy when applied for selectivity estimation, they are clearly not suitable for standard bar chart visualizations. Given our binned view definition, a possible binned bar chart representation of query Q is expressed as:

$$V_{i,b}: \texttt{SELECT A, F(M) FROM } D_B \texttt{ WHERE T GROUP BY A}$$
$$\texttt{NUMBER OF BINS b}$$

The deviation provided by a binned view $V_{i,b}$ is computed similar to that in Eq. 1. In particular, the comparison view is binned using a certain number of bins b and normalized into a probability distribution $P[V_{i,b}(D_B)]$. Similarly, the target view is binned using the same b and normalized into $P[V_{i,b}(D_Q)]$. Then the deviation $D(V_{i,b})$ is calculated as:

$$D(V_{i,b}) = dist(P[V_{i,b}(D_Q)], P[V_{i,b}(D_B)]) \tag{2}$$

2.3 View Processing Cost

Recall that in the absence of numerical dimensions, the number of candidate views N to be constructed is equal to $2 \times N$, where $N = |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$. In particular, $|\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$ queries are posed on the data subset D_Q to create the set of target views, and another $|\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$ queries are posed on the entire database D_B to create the corresponding set of comparison views. For each candidate non-binned view V_i over a numerical dimension A_j, the number of target and comparison binned views is equal to: $|\mathbb{M}| \times |\mathbb{F}| \times B_j$ each, where B_j is the maximum number of possible bins that can be applied on dimension A_j (i.e., number of binning choices). Hence, in the presence of $|\mathbb{A}|$ numerical dimensions, the total number of binned views grows to N_B which is simply calculated as:

$$N_B = 2 \times \sum_{j=1}^{|\mathbb{A}|} |\mathbb{M}| \times |\mathbb{F}| \times B_j \tag{3}$$

Furthermore, each pair of target and comparison binned views incur query execution time and deviation computation time. Query execution time is the time required to process the raw data to generate the candidate target and comparison binned views, where the cost for generating the target view is denoted as $C_t(V_{i,b})$, and that for generating the comparison view is denoted as $C_c(V_{i,b})$. Moreover, deviation computation time is the time required to measure the deviation between the target and comparison binned views, and is denoted as: $C_d(V_{i,b})$. Notice that this time depends on the employed distance function *dist*.

Putting it together, the total cost incurred in processing a candidate view V_i is expressed as:

$$C(V_i) = \sum_{b=1}^{B} C_t(V_{i,b}) + C_c(V_{i,b}) + C_d(V_{i,b}) \tag{4}$$

We note that the cost of computing deviation is negligible as compared to query execution cost, as it involves no I/O operations. Furthermore, for simplicity in the next sections we assume $C(V_{i,b}) = C_t(V_{i,b}) + C_c(V_{i,b})$. Therefore, Eq. 4 is reduced to:

$$C(V_i) = \sum_{b=1}^{B} C(V_{i,b}) \tag{5}$$

Hence, the total cost incurred in processing all candidate binned views is expressed as:

$$C = \sum_{i=1}^{N} C(V_i) \tag{6}$$

The goal of this study is to propose schemes that reduce the cost $C_t(V_{i,b})$ and $C_c(V_{i,b})$, which will consequently reduce the overall cost C of the solution.

3 Problem: Materialized View Selection

As mentioned in Sect. 2, the view recommendation process involves the generation of a huge number of the comparison and the target views. Particularly, these views are the result of executing their corresponding aggregate queries. Section 2.3 outlines how colossal the cost is for the binned view recommendation problem. However, we notice that for binned aggregate queries, the result of certain queries can be used to answer other queries. For instance, consider a view $V_{i,2} = (A, M, F, 2)$ can be answered from a number of other views such as $V_{i,4}$, $V_{i,6}$, $V_{i,8}$, etc., by performing aggregation on these views instead of the base table. We term this relationship as *dependency*. For instance, view $V_{i,2}$ depends on $V_{i,4}$, $V_{i,6}$ and $V_{i,8}$.

Definition: View Dependency: a binned view $V_{i,b}$ depends on another binned view $V_{i,b'}$, if $V_{i,b}$ can be answered using $V_{i,b'}$, where b' is a multiple of b i.e., $b' = xb$.

For any non-binned view V_i, all the possible binned views $V_{i,b}$ can be directly generated from the base table. Therefore, every $V_{i,b}$ at least depends on the base table, and at most depends on $\frac{B}{b} - 1$ other views $V_{i,b'}$. The dependency relationship between the candidates can be represented by a lattice. Figure 2 shows the lattice for a particular non-binned view V_i that can have a maximum of 8 bins. Each node in the lattice represents a binned view, e.g. node 5 is binned view $V_{i,5}$, while node 0 represents the base table. A view can be generated using any of its ancestors in the lattice. For instance, the ancestors of node 3 (i.e., $V_{i,3}$) are node 6 (i.e., $V_{i,6}$) and node 0 (i.e., base table).

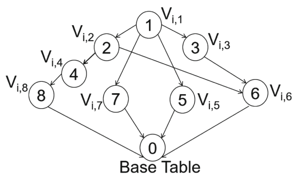


Fig. 2. Lattice for view V_i with $B = 8$

Every $V_{i,b'}$ is a candidate view that can be reused to generate some other views. Specifically, $V_{i,b}$ can be cached in the memory or stored on the disk for later reuse. However, because of the limited memory it is practical to store the view on the disk. Therefore, we propose to materialize the views that are later required to be reused. For instance, in Fig. 2, every view that is an ancestor of at least one other view is a candidate view to be materialized. A key problem is how to decide which views should be reused? The three possible options are:

1. *Reuse nothing:* This is the baseline case in which all the queries are answered from the base table. Consequently, this would incur the query processing time for each binned view from scratch.
2. *Reuse the whole lattice:* In this case all views should be materialized. This would reduce the query processing time of each binned view but the overall execution time of the solution will increase because it would include the additional cost of materializing the views.
3. *Reuse a set of views:* Choose an optimal set of views $\mathbb{T}$ to reuse and materialize them. This will incur the cost of materialization but reduce the overall cost of the solution because a number of queries will be answered from the materialized views instead of the base table.

The best option is to reuse a set of views, which has a possibility of reducing the overall cost. However, a cost benefit analysis between answering the views directly from the base tables vs. materializing the views and answering some views from those materialized ones is required. For that purpose, let $C_b(V_{i,b})$ be the cost of answering a binned view $V_{i,b}$ from the base table. Then in Eq. 5, the cost of finding the top-1 binned view $(C(V_i))$, for the non-binned view V_i, can be rewritten as:

$$C(V_i) = \sum_{b=2}^{\frac{L}{w}} C_b(V_{i,b}) \tag{7}$$

Notice $C(V_i)$ actually specifies the cost for option 1, where nothing is reused. For the other options, where reuse is involved, let $C_m(V_{i,b})$ be the cost of answering $V_{i,b}$ from a materialized view. Additionally, let the views be divided into two sets: (1) *Dependent Set*: the views that can be answered from $\mathbb{T}$ belong to the dependent set $\mathbb{P}$, and (2) *Independent Set*: the views that cannot be answered from $\mathbb{T}$ belong to the independent set $\mathbb{I}$. Particularly, the views in $\mathbb{I}$ need to

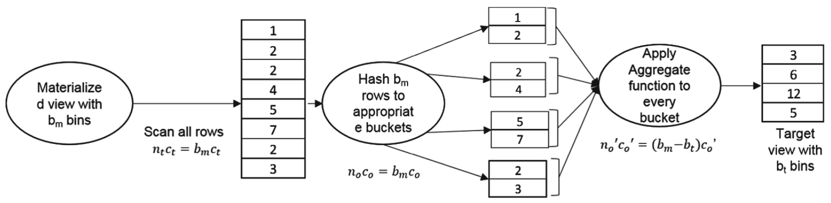

Materialized view with b_m bins
Scan all rows
$n_t c_t = b_m c_t$
1
2
2
4
5
7
2
3
Hash b_m rows to appropriate buckets
$n_o c_o = b_m c_o$
1
2
2
4
5
7
2
3
Apply Aggregate function to every bucket
$n_o' c_o' = (b_m - b_t) c_o'$
3
6
12
5
Target view with b_t bins

Fig. 3. Example of cost model for HashAggregate operator where $b_m = 8$ and $b = 4$

be answered from the base table. Let the cost of materializing a view $V_{i,b'}$ is $C_M(V_{i,b'})$, then Eq. 8 specifies the cost for option 3:

$$C(V_i) = \sum_{V_{i,b} \epsilon \mathbb{P}} C_m(V_{i,b}) + \sum_{V_{i,b} \epsilon \mathbb{T}} C_M(V_{i,b}) + \sum_{V_{i,b} \epsilon \mathbb{I}} C_b(V_{i,b}) \tag{8}$$

Definition: Materialized View Selection for View Recommendation: Given all the binned views $V_{i,b}$ for a non-binned view V_i, find a set $\mathbb{T}$ of views to materialize, which minimize the cost $C(V_i)$ of finding the top-1 binned view.

4 Methodology

Our proposed schemes in this section adapt and extend algorithms of materialized view selection towards efficiently solving the aggregate view recommendation problem.

4.1 mView: Greedy Approach

As explained int Sect. 2, the large number of possible binned views, makes the problem of finding the optimal binning for a certain view V_i highly challenging. An exhaustive brute force strategy is that given a certain non-binned view V_i, all of its binned views are generated and the utility of each of those views is evaluated. Consequently, the value of b that results in the highest utility is selected as the binning option for view V_i. However, this involves massive cost of processing all possible binned views.

In this work, we propose a novel technique *mView*, which instead of answering each query related to a view from scratch, reuses results from the already executed queries through view materialization. Particularly, *mView* maintains two sets of views; (1) $\mathbb{T}$: the views that are finalized to be materialized, (2) *Cand*: set of candidate views that can be added to $\mathbb{T}$ and consequently get materialized. The proposed technique *mView* adapts a greedy approach to determine $\mathbb{T}$ for materialization. Initially, *Cand* and $\mathbb{T}$ are empty. Then for a non-binned view V_i, a lattice as shown in Fig. 2 is constructed, using an adjacency list after identifying dependencies among the views. The search for the top-1 binned view starts from the binned view $V_{i,b}$ where $b = 1$. All of the views that are ancestors of $V_{i,b}$ in the lattice are added to the set *Cand*. Next, the benefit of materializing each view in *Cand* is computed.

We study in detail how to compute the benefit of materializing a view in the next paragraph. After benefit calculation, from the set *Cand*, a view V_m, which provides the maximum benefit is selected. V_m is added in $\mathbb{T}$ if it is not already in $\mathbb{T}$. Consequently, V_m is materialized and $V_{i,b}$ is generated from V_m. In next iteration *Cand* is set to empty again and the ancestors of the next binned view are added to *Cand*. This process goes on until all of the $V_{i,b}$ have been generated. Initial experiments show this approach reduced cost of Linear search by 25–30% Clearly, for this technique to work efficiently, a cost model is required to estimate the benefit of materializing views without actual materialization. Therefore, next we define that cost and benefit model.

Cost Benefit Analysis. As mentioned earlier, to decide which views are the best candidates for materialization, the cost and benefit of materialization needs to be analyzed. Specifically, we use processing time as our cost metric to measure performance of the schemes. In the linear cost model, the time to answer a query is taken to be equal to the space occupied by the underlying data from which the query is answered [1,7]. In this work, the same model is adopted with some modifications. Assume that the time to answer the aggregate query Q is related to two factors; (1) the number of tuples of the underlying view from which Q is answered, which is actually the number of bins of the ancestor view, and (2) the amount of aggregation required to answer Q. Normally, a relational DBMS uses HashAggregate as query execution plan for group-by queries. Particularly, in this study PostgreSQL is used as backend database, which uses HashAggregate as query execution plan for group-by queries. Hence, the cost model used by the query optimizer, particularly PostgreSQL consists of a vector of five parameters to predict the query execution time [14]; (1) Sequential page cost (c_s), (2) Random page cost (c_r), (3) CPU tuple cost (c_t), (4) CPU index tuple cost (c_i), and (5) CPU operator cost (c_o). The cost C_{HA} of the HashAggregate operator in a query plan is then computed by a linear combination of c_s, c_r, c_t, c_i, and c_o:

$$C_{HA} = n_s c_s + n_r c_r + n_t c_t + n_i c_i + n_o c_o$$

Where the values $n = (n_s, n_r, n_t, n_i, n_o)^{\mathbb{T}}$ represent the number of pages sequentially scanned, the number of pages randomly accessed, and so forth, during the execution.

Generally, for estimating cost of an operator, the values in vector n are estimated. However, in our case, the already known number of rows of a materialized view (i.e., number of bins of that view) and target view can be used for vector n. For instance, Fig. 3 shows the steps of the HashAggregate operator for generating a view with 4 bins from a view with 8 bins, and the cost incurred. Specifically, The operation of generating a view with b bins from a view with b_m bins has the following parameters:

– $n_s c_s$ & $n_r c_r$: c_s and c_r are the I/O costs to sequentially access a page and randomly access a page, while n_s and n_r are the number of sequentially and randomly accessed pages respectively. Generally, size of a page is 8 KB. Consequently, n_s depends on the page size, size of each row (let it be r), and the number of rows read, which is equal to the number of bins of the materialized view, i.e., $n_s = \frac{8\,\text{KB}}{r \times b_m}$. Furthermore, c_s and c_r depends on whether the data is fetched from the disk or it is already in cache. Particularly, this cost is negligible for the later case and that is the case in our model.

- $n_t c_t$: c_t is the cost of scanning each row and n_t is the number of rows scanned, which is equal to the number of bins of the materialized view, i.e., $n_t = b_m$.
- $n_i c_i$: c_i is the cost to place the row in a bucket (bin) using hashing and n_i is the number of rows hashed, which is equal to the number of bins of the materialized view, i.e., $n_i = b_m$.
- $n_o c_o$: c_o is the cost to perform aggregate operation such as sum, count etc., and n_o is the number of aggregate operations performed. If $V_{i,b}$ is answered from V_{i,b_m}, then there are b buckets and each bucket will require $\frac{b_m}{b} - 1$ aggregate operations, i.e., $n_o = b(\frac{b_m}{b} - 1) = b_m - b$.

Therefore, the cost of HashAggregate operator C_{HA} is:

$$C_{HA} = n_t c_t + n_i c_i + n_o c_o$$
$$C_{HA} = b_m c_t + b_m c_i + (b_m - b) c_o$$
$$C_{HA} = b_m (c_t + c_i + c_o) + b(-c_o)$$

The costs c_t, c_i, and c_o remain same for all queries. Therefore, we replace them with simple constants c and c' such that: $c = c_t + c_i + c_o$ and $c' = -c_o$. Hence,

$$C_{HA} = b_m \times c + b \times c'$$

Therefore, the cost of generating $V_{i,b}$ from materialized view V_{i,b_m} is:

$$C_m(v_{i,b}) = b_m \times c + b \times c' \tag{9}$$

Where c and c' are learnt through multi-variable linear regression. Consequently, the benefit of materializing a view V_{i,b_m} is computed by adding up the savings in the query processing cost for each dependent view $V_{i,b}$ over answering $V_{i,b}$ from the base table and subtracting the cost of materialization of V_{i,b_m}.

$$\mathbb{B}(V_{i,b_m}) = \sum_{V_{i,b} \in \mathbb{P}} [(C_b(V_{i,b}) - C_m(V_{i,b}))] - C_M(V_{i,b_m}) \tag{10}$$

In this section, we listed the details of our proposed technique *mView* for the exhaustive search, which is also called *Linear* search. When this scheme is applied to a non-binned view V_i, it results in a top-1 binned view, this is termed has horizontal search. Furthermore, applying this to every non-binned view, their corresponding top-1 binned views are identified and from there top-k views can be easily recommended, this is termed as vertical search. In our experiments, we differentiate between horizontal and vertical search and the scheme applied to each direction.

4.2 Materialized Views with MuVE

In [3,4], we argue that the deviation based utility metric falls short in completely capturing the requirements of numerical dimensions. Hence, a hybrid multi-objective utility function was introduced, which captures the impact of numerical dimension attributes in terms of generating visualizations that have: (1) interestingness ($D(V_{i,b})$): measured using the deviation-based metric, (2) usability ($S(V_{i,b})$): quantified via the relative bin width metric, and (3) accuracy ($A(V_{i,b})$): measured in terms of Sum Squared Error (SSE). The proposed multi-objective utility function, was defined as follows:

$$U(V_{i,b}) = \alpha_D \times D(V_{i,b}) + \alpha_A \times A(V_{i,b}) + \alpha_S \times S(V_{i,b}) \tag{11}$$

Parameters α_D, α_A and α_S specify the weights assigned to each objective, such that $\alpha_D+\alpha_A+\alpha_S = 1$. Furthermore, to efficiently navigate the prohibitively large search space *MuVE* scheme was proposed, which used an incremental evaluation of the multi-objective utility function, where different objectives were computed progressively. In this section, we discuss how to achieve benefits of both the schemes, *mView* and *MuVE*.

Selecting $\mathbb{T}$ while using *MuVE* as search strategy is non-trivial, because of the trade-off between *MuVE* and *mView*. In the *MuVE* scheme, the benefit of cost savings comes from the pruning of many views and utility evaluations. A blind application of greedy view materialization, as in *mView*, may result in materialization of views that gets pruned because of the *MuVE's* pruning scheme. The idea here is to estimate which views *MuVE* will eliminate and exclude those views from the set of candidate views to materialize. To address this issue, we introduce a penalty metric, which is added to the benefit function. Therefore, a candidates view V_{i,b_m}, which has high certainty (represented as $CE(V_{i,b_m})$) of getting pruned by *MuVE* gets a high reduction in its benefit of materialization. Particularly, a view gets pruned due to either of the two factors; (1) short circuit of deviation objective, the certainty of this pruning is represented as $CE_D(V_{i,b_m})$, and (2) early termination, certainty of getting early terminated is represented as $CE_E(V_{i,b_m})$. The certainty factor $CE(V_{i,b_m})$ is the sum of the certainty of pruning deviation evaluation ($CE_D(V_{i,b_m})$) and certainty of getting early terminated ($CE_E(V_{i,b_m})$).

$$CE(V_{i,b_m}) = CE_D(V_{i,b_m}) + CE_E(V_{i,b_m})$$

Therefore, the benefit of materializing a view in Eq. 10 is updated as:

$$\mathbb{B}(V_{i,b_m}) = \sum_{V_{i,b} \in \mathbb{P}} [C_b(V_{i,b}) - C_m(V_{i,b})] - [CE(V_{i,b_m}) \times C_M(V_{i,b_m})] \tag{12}$$

The certainty of pruning deviation computation depends on the ratio of α_A and α_D. *MuVE* uses a priority function to determine which objective to evaluate first, in other words *MuVE* tries to prune the objective, which is not evaluated first. According to that function if α_A is greater than α_D there is a chance of pruning the deviation objective. We are interested in pruning deviation evaluation as it is the only objective that involves execution queries for target and comparison views.

$$CE_D(V_{i,b}) = \left\{ \begin{matrix} 0 \ for \ \frac{\alpha_A}{\alpha_D} < 1 \\ \frac{\alpha_A}{\alpha_D} \times 10 \ for \ \frac{\alpha_A}{\alpha_D} \geq 1 \end{matrix} \right\} \tag{13}$$

The certainty of early termination depends on α_S and b, higher value of α_S or b means the chance of getting early termination is high.

$$CE_E(V_{i,b}) = \left\{ \begin{matrix} 0 \ for \ \alpha_S < 0.5 \\ \alpha_S \times \frac{b}{L} \ for \ \alpha_S \geq 0.5 \end{matrix} \right\} \tag{14}$$

5 Experimental Evaluation

5.1 Experimental Testbed

We perform extensive experimental evaluation to measure the efficiency of top-k view recommendation strategies presented in this paper. Here, we present the different parameters and settings used in our experimental evaluation.

Setup: We built a platform for recommending visualizations, which extends the SeeDB codebase [13] to support view materialization based schemes presented in this paper. Our experiments are performed on a Corei7 machine with 16 GB of RAM. The platform is implemented in Java and PostgreSQL is used as the backend DBMS.

Schemes: We investigate the performance of the different combinations of the vertical and horizontal search strategies presented in [3] with *mView* proposed in this paper. Our naming convention for those combinations is represented as: *SearchH-SearchV*, where *SearchH* denotes the search strategy employed for horizontal search, whereas *SearchV* is the one for the vertical search. This leads to the following combinations: *Linear-Linear*, *MuVE-Linear*, and *MuVE-MuVE* as baseline schemes and *mView(Linear-Linear)*, *mView(MuVE-Linear)*, and *mView(MuVE-MuVE)* as proposed schemes.

Data Analysis: As in [13], we assume a data exploration setting in which a multi-dimensional dataset of diabetic patients[1] is analyzed. The DIAB dataset has 9 attributes and 768 tuples. The independent numeric attributes of the dataset are used as dimensions (e.g., age, BMI, etc.), whereas the observation attributes are used as measures (insulin level, glucose concentration, etc.). In our default setting, we select 3 dimensions, 3 measures, and 3 aggregate functions, which results in a maximum of 2961 possible views. In the analysis, all the α values are in the range $[0 - 1]$, where $\alpha_D + \alpha_A + \alpha_S = 1$. In the default setting, $\alpha_D = 0.2$, $\alpha_A = 0.2$, $\alpha_S = 0.6$, $k = 5$, and euclidean distance is used for measuring deviation, unless specified otherwise.

[1] https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes.

Performance: We evaluate the efficiency and effectiveness of the different recommendations strategies in terms of two factors: (1) *Cost:* As mentioned in Sect. 3, the cost of a strategy is the total cost incurred in processing all the candidate binned views. We use wall clock time to measure the different components included in that cost namely, query execution time of target and comparison views, deviation computation time, and accuracy evaluation time. (2) *Relative Difference:* The ratio between cost of baseline schemes and the $mView$ based schemes, i.e., $\frac{Cost of baseline - Cost of mView Scheme}{Cost of baseline scheme}$. Each setting is executed 10 times and then average is taken as the cost incurred.

5.2 Experiments

In the following experiments, we evaluate the performance of our technique $mView$ under different parameter settings. As explained in Sect. 4 that $mView$ scheme is used in combination with the baseline Linear scheme and optimized *MuVE* scheme. Additionally it was also mentioned that the blind materialization of views while using *MuVE* search strategy may not be the optimal solution. Therefore, for *mView(MuVE-MuVE)* and *mView(MuVE-Linear)* schemes an heuristic based method was proposed to predict the expected early termination and short circuit point. Figures 4 and 6 show the impact on cost, while Figs. 5 and 7 quantifies the percentage improvement achieved in terms of relative difference using the view materialization scheme.

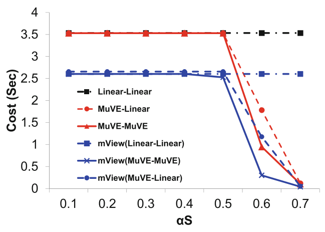


Fig. 4. Impact of α_A and α_S on cost, while $\alpha_D = 0.2$

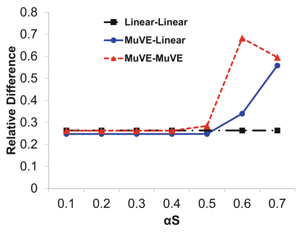


Fig. 5. Impact of α_A and α_S on relative difference, while $\alpha_D = 0.2$

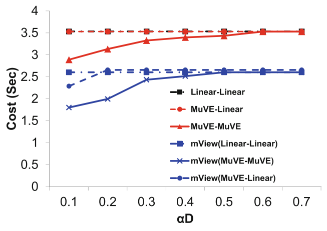


Fig. 6. Impact of α_A and α_D on cost, while $\alpha_S = 0.2$

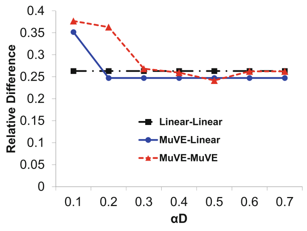


Fig. 7. Impact of α_A and α_D on relative difference, while $\alpha_S = 0.2$

In Figs. 4 and 5, α_D is set to constant 0.2 while α_A and α_S are changing. In particular, as shown in the figures, α_S is increased, while α_A is implicitly decreased and is easily computed as $\alpha_A = 1 - \alpha_D - \alpha_S$. Figure 4 shows that cost *mView(Linear-Linear)* is less than the baseline scheme *Linear-Linear*. This is because *mView* chooses such a set of views to materialize that saves aggregation time by generating them from the materialized views. Furthermore, Fig. 5 shows *mView(Linear-Linear)* reduces the cost by almost 30% as compared to the *Linear-Linear* scheme. Figure 4 also shows that using our proposed heuristic in *mView(MuVE-MuVE)* and the incremental view materialization of *mView*, the cost is further reduced. This is due to the reason that we avoided the unnecessary materialization of views which are eventually pruned by *mView(MuVE-MuVE)*. Furthermore, Fig. 5 shows *mView(MuVE-MuVE)* reduces the cost by almost 70% as compared to *MuVE-MuVE* at $\alpha_S = 0.6$.

In Figs. 6 and 7, α_S is set to constant 0.2 while α_A and α_D are changing. Figure 6 clearly shows that *mView* based three schemes have less cost compared to the other three schemes. The difference in cost for the *mView(MuVE-MuVE)* scheme is more than 30% at $\alpha_D = 0.1$ as shown in Fig. 7.

6 Conclusions

In this paper we presented a novel technique *mView* for recommending top-k binned aggregate data visualizations. The proposed scheme reuses the already executed views through materialization and answering the later queries from the materialized views. We defined a cost benefit model to decide which views can be reused later. We also proposed a heuristic based approach to predict the expected early termination and short circuit for *MuVE* based schemes. Our experimental results show that employing the *mView* technique for both *Linear* and *MuVE* based schemes offers significant reduction in terms of data processing costs.

References

1. Baralis, E., Paraboschi, S., Teniente, E.: Materialized views selection in a multidimensional database. In: VLDB, pp. 156–165 (1997)
2. Chaudhuri, S., Krishnamurthy, R., Potamianos, S., Shim, K.: Optimizing queries with materialized views. In: ICDE, pp. 190–200 (1995)
3. Ehsan, H., Sharaf, M.A., Chrysanthis, P.K.: MuVE: efficient multi-objective view recommendation for visual data exploration. In: ICDE, pp. 731–742 (2016)
4. Ehsan, H., Sharaf, M.A., Chrysanthis, P.K.: Efficient recommendation of aggregate data visualizations. IEEE Trans. Knowl. Data Eng. **30**(2), 263–277 (2018)
5. Gupta, H., Mumick, I.S.: Selection of views to materialize in a data warehouse. IEEE Trans. Knowl. Data Eng. **17**(1), 24–43 (2005)
6. Halevy, A.Y.: Answering queries using views: a survey. VLDB J. **10**(4), 270–294 (2001)
7. Harinarayan, V., Rajaraman, A., Ullman, J.D.: Implementing data cubes efficiently. In: SIGMOD, pp. 205–216 (1996)

8. Kandel, S., Parikh, R., Paepcke, A., Hellerstein, J.M., Heer, J.: Profiler: integrated statistical analysis and visualization for data quality assessment. In: AVI, pp. 547–554 (2012)
9. Key, A., Howe, B., Perry, D., Aragon, C.R.: VizDeck: self-organizing dashboards for visual analytics. In: SIGMOD, pp. 681–684 (2012)
10. Mafrur, R., Sharaf, M.A., Khan, H.A.: DiVE: diversifying view recommendation for visual data exploration. In: CIKM, pp. 1123–1132 (2018)
11. Srivastava, D., Dar, S., Jagadish, H.V., Levy, A.Y.: Answering queries with aggregation using views. In: VLDB, pp. 318–329 (1996)
12. Stolte, C., Tang, D., Hanrahan, P.: Polaris: a system for query, analysis, and visualization of multidimensional relational databases. IEEE Trans. Vis. Comput. Graph. **8**(1), 52–65 (2002)
13. Vartak, M., Rahman, S., Madden, S., Parameswaran, A.G., Polyzotis, N.: SEEDB: efficient data-driven visualization recommendations to support visual analytics. PVLDB **8**(13), 2182–2193 (2015)
14. Wu, W., Chi, Y., Zhu, S., Tatemura, J., Hacigümüs, H., Naughton, J.F.: Predicting query execution time: are optimizer cost models really unusable? In: ICDE, pp. 1081–1092 (2013)