Efficient Recommendation of Aggregate Data Visualizations

Humaira Ehsan, Mohamed A. Sharaf, and Panos K. Chrysanthis, *Senior Member, IEEE*

Abstract—Data visualization is a common and effective technique for data exploration. However, for complex data, it is infeasible for an analyst to manually generate and browse all possible visualizations for insights. This observation motivated the need for automated solutions that can effectively recommend such visualizations. The main idea underlying those solutions is to evaluate the utility of all possible visualizations and then recommend the top-k visualizations. This process incurs high data processing cost, that is further aggravated by the presence of numerical dimensional attributes. To address that challenge, we propose novel view recommendation schemes, which incorporate a hybrid multi-objective utility function that captures the impact of numerical dimension attributes. Our first scheme, Multi-Objective View Recommendation for Data Exploration (MuVE), adopts an incremental evaluation of our multi-objective utility function, which allows pruning of a large number of low-utility views and avoids unnecessary objective evaluations. Our second scheme, upper MuVE (uMuVE), further improves the pruning power by setting the upper bounds on the utility of views and allowing interleaved processing of views, at the expense of increased memory usage. Finally, our third scheme, Memory-aware uMuVE (MuMuVE), provides pruning power close to that of uMuVE, while keeping memory usage within a pre-specified limit.

Index Terms—Data exploration, visual analytics, aggregate views, view recommendation

1 Introduction

Recommending data visualizations that reveal new and valuable insights is a challenging problem, which has been the focus of many research approaches (e.g., [19], [20], [26], [30], [31]). The main idea underlying those approaches is to automatically generate all possible *aggregate views* of data, and recommend the *top-k* views that result in interesting visualization, where the interestingness of a visualization is quantified according to some utility function. Recent work provides strong evidence that a *deviation-based* formulation of utility is able to provide analysts with interesting visualizations that highlight some of the particular trends of the analyzed datasets [30], [31].

While the deviation-based notion of utility has been shown to be effective in recommending views with categorical dimensional attributes, in this work we argue that it falls short in capturing the requirements of numerical dimensions. Particularly, in the presence of such numerical dimensions, *binned aggregation* is required to group the numerical values along a dimension into adjacent intervals [10], [17]. Given the large number of options for binning a numerical dimension, it is expected that different binning configuration will result in different deviations, and in turn, different levels of interestingness from the analyst point of view. For instance, in a view with small number of bins, interesting insights are expected to remain hidden under a smooth and coarse visual representation. Meanwhile, in a view that contains a large number of bins, insights might go unnoticed in a cluttered or sparse visualization. To illustrate the impact of binning on numerical dimensions, consider the following example:

Example 1. Consider a data analyst trying to gain insights into the special factors that led the Golden State Warriors (GSW) basketball team to win the 2015 NBA championship. Consequently, the analyst uses the 2015 NBA players statistics database [4] to compare the GSW team to the other teams in the league. Particularly, the analyst poses a query:

```
Q: SELECT * FROM players WHERE team=GSW;
```

which returns the statistics for all the players on the GSW team. Such statistics include different dimensions (e.g., age, number of games played, minutes played, etc.), and different measures (e.g., player efficiency rating, 3-point attempt rate, etc.). To recommend interesting bar chart visualizations, different SQL aggregate functions are applied on the views resulting from all the possible pairwise combinations of dimensions and measures, then the most interesting views are presented to the analyst. Fig. 1 shows one particular view defined on the dimension `minutes played (MP)` and the measure `3-point attempt rate (3PAr)`. Such view is equivalent to:

```
V: SELECT MP, SUM (3PAr) FROM players WHERE team=GSW GROUP BY MP;
```

Meanwhile, generating the same view of the entire database of all players (i.e., without the `WHERE team=GSW` clause), results in the visualization shown in Fig. 2. At first glance, comparing the two views fails to reveal any insights about the GSW team. However, binning the two views, as shown in Fig. 3, reveals some very interesting observation. Particularly, Fig. 3 shows that for all NBA

- *H. Ehsan and M.A. Sharaf are with the School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane QLD 4072, Australia. E-mail: {h.ehsan, m.sharaf}@uq.edu.au.*
- *P.K. Chrysanthis is with the Department of Computer Science, School of Computing and Information, University of Pittsburgh, Pittsburgh, PA 15260. E-mail: panos@cs.pitt.edu.*

Manuscript received 2 Mar. 2017; revised 12 Sept. 2017; accepted 3 Oct. 2017. Date of publication 24 Oct. 2017; date of current version 9 Jan. 2018. (Corresponding author: Humaira Ehsan.) Recommended for acceptance by N. Zhang. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TKDE.2017.2765634

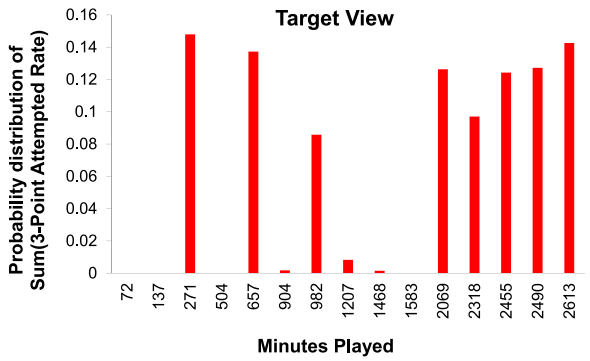


Fig. 1. View on players of the GSW team (target view).

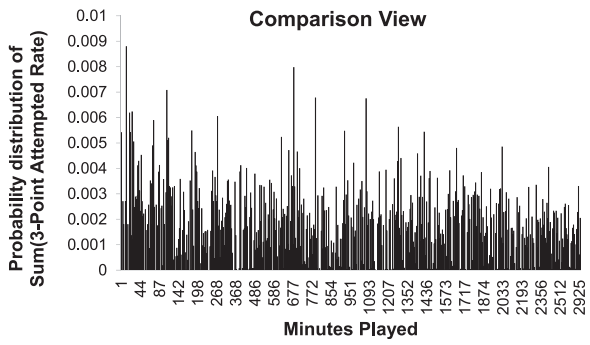


Fig. 2. View on all players in the 2015 NBA (comparison view).

players, the 3PAr decreases as they play more games. The intuitive explanation is that the fatigue incurred from playing more games can affect their fitness and reduce their 3PAr. However, for the GSW players, that pattern significantly deviates from that general pattern. As Fig. 3 shows, the GSW players who spent more time on the field still achieve very high 3PAr. In fact, their 3PAr is almost 4 times that of the other players. Clearly, that observation reflects the fitness and consistency of the GSW players, which might distinguish them from other players in the league, and can shed some light into understanding their championship win.

As the above example shows, choosing the right binning is essential in the process of extracting insights from the data, whether that process is performed manually or analytically. On the one hand, a good binning allows to reduce both the clutter and sparsity in the generated visualizations, which makes them easy to use by the analyst to manually extract insights [7], [11]. On the other hand, a good binning also allows to group similar data together (e.g., group players according to their MP), so that the special features of each group (e.g., 3PAr) is aggregated and emphasized, which in turn allows quantitative metrics, such as deviation, to capture the interesting patterns exhibited by those features. We note, however, that choosing the right binning for each visualization is a non-trivial task. The benefits, as well as the challenges, of binning numerical dimensions are well-recognized in the literature, especially in the context of histogram construction for the purpose of selectivity estimation and query optimization (e.g., [10], [17], [18]). Such histograms provide a concise summary of the underlying data distribution of an attribute, where the accuracy of that summarization is dependent on the employed binning strategy. Similarly, in bar chart visualizations, which is the focus of this paper, the overall utility of a visualization is dependent on the

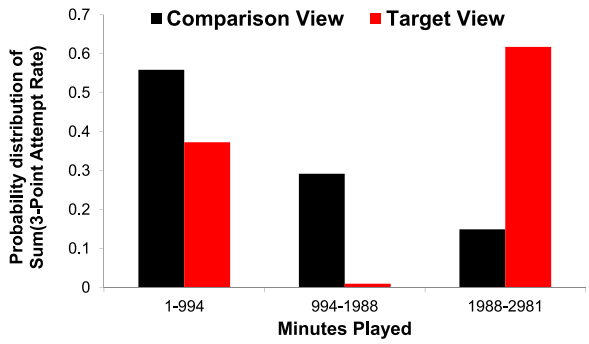


Fig. 3. Binned target view (i.e., GSW team) and comparison view (i.e., all NBA teams).

underlying binning. Consequently, the applicability of the simple deviation-based notion of utility becomes very limited in the presence of numerical dimension attributes.

To address such limitation, we introduce a novel hybrid multi-objective utility function, which captures the impact of numerical dimension attributes in terms of generating visualizations that are: 1) *interesting*, 2) *usable*, and 3) *accurate*. Clearly, combining these often conflicting objectives dramatically expands the search space of possible visualizations (i.e., aggregate views). Moreover, it significantly increases the processing time incurred to asses the overall utility of each view, which is assembled from the utility values of each of the three objectives listed above.

Our earlier work demonstrated that using specific features of the view recommendation problem allows to efficiently navigate the prohibitively large search space of possible views [13]. In this work, we present a novel suite of search schemes for efficient recommendation of top-k aggregate data views. The main idea underlying our first scheme *Multi-Objective View Recommendation for Data Exploration* (*MuVE*) is to use an incremental evaluation of the multi-objective utility function, where different objectives are computed progressively. Our results in [13] show that MuVE is able to prune a large number of unnecessary views, and in turn reduces the overall processing time for recommending the top-k views. However, that achieved pruning power is highly dependent on the order in which those views are presented to MuVE and might often limit its performance gains. To address that limitation, we propose our second scheme *upper MuVE* (*uMuVE*), in which the goal is to provide a flexible navigation of the search space so that high-utility views are discovered earlier. Particularly, uMuVE is based on setting upper bounds on the utility of each possible view, which is then exploited to effectively guide the search process by means of interleaving the evluation of the different objectives offered by the different views. Due to that interleaved processing, at any point of time, uMuVE would typically have multiple views under consideration, which requires significant amount of memory for storing their data. This motivated us to propose our third scheme *Memory-aware uMuVE* (*MuMuVE*), which aims to provide a pruning power close to that of uMuVE under some memory usage constraints. We also included set of experiments to demonstrate efficiency of our optimized view recommendation techniques.

In summary, the main contributions of this work are as follows:

- We analyze the problem of recommending visualizations in the presence of numerical dimension

TABLE 1
Summary of Symbols

Symbol	Description
D_B	Database
Q	Input query
T	Predicates in input query
D_Q	Data returned by input query Q
A	Dimension attribute
M	Measure attribute
F	Aggregate function
V_i	Aggregate view i
$V_{i,b}$	Aggregate view i with b bins
$D(V_{i,b})$	Interestingness of view $V_{i,b}$
$S(V_{i,b})$	Usability of view $V_{i,b}$
$A(V_{i,b})$	Accuracy of view $V_{i,b}$
$U(V_{i,b})$	Utility of view $V_{i,b}$
$\alpha_D\ \alpha_A\ \alpha_S$	Weights of D, A and S

attributes and formulate it in terms of a novel hybrid multi-objective utility function.
- We propose novel search algorithms MuVE, uMuVE and MuMuVE, which are particularly optimized to leverage the specific features of the view recommendation problem.
- We conduct extensive experimental evaluation using real datasets, which illustrate the benefits achieved by our proposed algorithms.

The rest of our paper is organized as follows. In Section 2, we describe the background for our problem. We then formally describe our multi-objective utility function and problem statement in Section 3, The proposed search algorithms are discussed in detail in Section 4 and Section 5. We present our testbed and experiments on real data in Sections 6 and 7. Finally, we discuss related work in Section 8 and conclude in Section 9.

2 Preliminaries

2.1 View Recommendation

As in our example above, 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:

Q: `SELECT * FROM` D_B `WHERE T;`

In Q, T specifies a combination of predicates, which selects a portion of D_B for visual analysis (e.g., `team = GSW`). 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 [19], [20], [28], [30], [31], 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 (all symbols are summarized in Table 1). As in [31], we consider aggregate views that perform a single-attribute group-by and aggregation on D_Q. A possible view V_i of the example query Q above would be expressed as:

V_i: `SELECT A, F(M) FROM` D_B `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., [19], [20], [30], [31]). Among those metrics, recent case studies have shown that a *deviation-based* metric is able to provide analysts with interesting visualizations that highlight some of the particular trends of the analyzed datasets [31].

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 versus 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 $P[V_i(D_B)]$. Consider an aggregate view $V = (A, M, F)$. The result of that view can be represented as the set of tuples: $< (a_1, g_1), (a_2, g_2), \ldots, (a_t, g_t) >$, where t is the number of distinct values (i.e., groups) in attribute A, a_i is the ith group in attribute A, and g_i is the aggregated value $F(M)$ for the group a_i. Hence, V is normalized by the sum of aggregate values $G = \sum_{p=1}^{t} g_p$, resulting in the probability distribution $P[V] = < \frac{g_1}{G}, \frac{g_2}{G}, \ldots, \frac{g_t}{G} >$. For instance, for the comparison view shown in Fig. 3, Table 2 illustrates the groups (Minutes played), aggregate values (Sum 3-PAR) and the computation of its probability distribution.

TABLE 2
Computing the Probability Distribution of the Comparison View Shown in Fig. 3

Minutes Played	Sum (3-PAR)	Pdf Sum(3-PAR)
$a_1 : 1 - 994$	$g_1 : 103.32$	$\frac{g_1}{G} = \frac{103.32}{184.87} = 0.56$
$a_2 : 994 - 1988$	$g_2 : 53.97$	$\frac{g_2}{G} = \frac{53.97}{184.87} = 0.29$
$a_3 : 1988 - 2981$	$g_3 : 27.58$	$\frac{g_3}{G} = \frac{27.58}{184.87} = 0.15$
Sum of aggregate values $(G) = 184.87$		

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

$$D(V_i) = dist(P[V_i(D_Q)], P[V_i(D_B)]). \quad (1)$$

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*) [31]. 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. Thus, several techniques have been proposed for optimizing the processing time incurred in recommending visualizations, which are orthogonal to the optimizations proposed in this work to address the impact of numerical dimensions, which is described next.

2.2 Numerical Dimensions

In this paper, we mainly focus on the problem of recommending visualizations in the presence of numerical dimension attributes. While numerical dimension attributes (e.g., age, height, etc.) are abundant in scientific and commercial databases, current visualization recommendation techniques tend to mostly overlook such numerical dimensions, and rather focus on the categorical ones. In the presence of numerical dimensions, *binned aggregation* is typically required so that to group the numerical values along a dimension into adjacent intervals over the range of values covered by that dimension [10], [17]. Accordingly, binning of numerical dimensions poses several non-trivial challenges in terms of recommending visualizations that are not only interesting, but also *accurate* and *usable*. Particularly, in addition to being interesting (i.e., highly deviated from the general data D_B), recommended visualizations are expected to be accurate (i.e., minimize the amount of *error* between the aggregated view V_i and its corresponding dataset D_Q) and usable (i.e., minimize the amount of *clutter* in view V_i). For instance, while the target and comparison views shown in Figs. 1 and 2 are highly accurate (no binning applied), they are also barely usable because of high clutter or high sparsity, which translates into missing out on revealing interesting insights.

As mentioned earlier, the benefits, as well as the challenges, of binning numerical dimensions are well-recognized in the literature, especially in the context of histogram construction (e.g., [10], [17], [18]), anomaly detection (e.g., [29]), and data visualization (e.g., [19], [23]). For instance, binning (also know as bucketing) is an essential step in constructing histograms over numerical attributes for the purpose of selectivity estimation and query optimization. Deciding the optimal bin width for histograms has been intensively studied in the statistics literature, where several *model-based* approaches have been proposed [10]. In contrast, the database literature mostly takes a *model-free* approach, considering the dataset currently stored in the database as the only data of interest. (We refer the reader to [10], [17], for comprehensive surveys on that topic.) In this work, we adopt the same approach and expand on existing model-free methods, as discussed next.

2.3 Binned Views

To enable the incorporation and recommendation of visualizations that are based on continuous numerical dimensions, in this work we introduce 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:

Definition 1 (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. 3 shows a binned view, in which the number of bins $b = 3$ and the bin width $w = 994$.

We note that our 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:

$V_{i,b}$:
```
SELECT A, F(M) FROM D_B WHERE T
GROUP BY A
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

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

Clearly, the deviation $D(V_{i,b})$ is bounded by a maximum value D_M. That value D_M is achieved when for each group a_i, the corresponding value $\frac{g_i}{G}$ in $P[V_{i,b}(D_B)]$ or $P[V_{i,b}(D_Q)]$ is zero. Therefore, to normalize the deviation, $D(V_{i,b})$ is divided by D_M.

Clearly, for a binned view such as view $V_{i,b}$ defined above, its *usability* depends on the visual quality i.e., how clearly the visualization reveals the structure within the data. Inversely, clutter is defined by the crowdedness that obscure the structure within the visualization [7]. A number of metrics have been proposed to measure clutter for various types of visualizations, such as the number of data points displayed, data density (number of data points/number of pixels), data to ink ratio and lie factor [7], [11]. All of these metrics basically quantify the amount of content displayed on screen as a measure of clutter. Similarly, for bar chart visualizations, clutter occurs due to the large number of bins in a visualization. Consequently, we simply define usability as the inverse of clutter, which is captured as follows:

$$S(V_{i,b}) = \frac{1}{b} = \frac{w}{L}, \quad (3)$$

where b is the number of bins, w is the width of each bin and L is the range of the dimension attribute. $S(V_{i,b})$ is in the range $[0, 1]$, such that $S(V_{i,b}) = 1$ indicates highest quality.

Furthermore, a binned view $V_{i,b}$ is obviously a summarized approximation of the corresponding non-binned view V_i. Thus, it is essential to measure the (in)accuracy provided by $V_{i,b}$. To achieve this, consider a non-binned view V_i, which is defined as (A, M, F). Further, and without loss of generality, assume A is an ordered integer attribute. As described in the previous section, the result of that view can be represented as

the set of tuples: $<(a_1, g_1), (a_2, g_2), \ldots, (a_j, g_i), \ldots, (a_t, g_t)>$, where t is the number of distinct values (i.e., groups) in attribute A. A binned view $V_{i,b}$ provides a concise approximate representation of V_i based on partitioning the ordered attribute A into b bins. Particularly, each bin I_x consists of a start and end point, $I_x = (s_x, e_x)$, and a value $\hat{g}_x$, which represents the aggregated value of the measure M over all the values of dimension A in the range of I_x. That is, $V_{i,b} = <(I_1, \hat{g}_1), (I_2, \hat{g}_2), \ldots, (I_x, \hat{g}_x), \ldots, (I_b, \hat{g}_b)>$, where $b \ll t$.

This data reduction implies approximation errors in the estimation of the original non-binned aggregate values, where the error incurred by that approximation increases with decreasing the number of bins b. There are number of metrics that have been used for measuring that kind of (in) accuracy such as Sum Squared Error, Sum-Absolute-Error, Sum-Absolute-Relative-Error and Maximum-Absolute Relative-Error [9]. In this work, we adopt Sum Squared Error (SSE) metric, which has also been widely employed by the database community to measure the accuracy of frequency histograms for the purpose of query optimization (e.g., [9], [18]). Applying SSE metric for general aggregate views is straightforward. In particular, the aggregate measure corresponding to any dimension value in the contiguous range $s_x, s_x + 1, ..., e_x$ is approximated using a single representative value g'_x, which is computed as $\frac{\hat{g}_x}{n_x}$, where $n_x = e_x - s_x + 1$ (i.e., the number of distinct values in bin b_x). Accordingly, each $g_j \in I_x$ is estimated as $g'_j = g'_x$. Hence, the SSE provided by $V_{i,b}$, denoted as $E(V_{i,b})$, is computed as $E(V_{i,b}) = \sum_{p=1}^{t} (g_p - g'_p)^2$ and the relative SSE is computed as $R(V_{i,b}) = \sum_{p=1}^{t} \frac{(g_p - g'_p)^2}{g_p^2}$. Accordingly, the accuracy of a view $V_{i,b}$ is simply computed as

$$A(V_{i,b}) = 1 - \frac{R(V_{i,b})}{t}. \quad (4)$$

The computed $A(V_{i,b})$ is in the range $[0, 1]$, such that $A(V_{i,b}) = 1$ indicates maximum accuracy (i.e., zero error).

Clearly, incorporating the different metrics listed above further complicates the problem of finding the top-k recommended visualizations. This is mainly due to the different binning options, which in turn leads to an increase in the number of candidate visualizations. Next, we formally define the problem of multi-objective view recommendation in the presence of binned views, as well as the costs incurred in solving such problem.

3 Problem Definition

In a nutshell, the goal of this work is to recommend the *top-k* bar chart visualizations of the results of query Q according to some utility function. When all dimension attributes are categorical, such goal simply boils down to recommending the *top-k* interesting views based on the deviation metric [30], [31], as described in Section 2.1. However, that simple notion of utility falls short in capturing the impact of numerical dimensions. In particular, the presence of numerical dimensions introduces additional factors that impact the utility gained from a recommended view. In our proposed schemes, we employ a novel hybrid multi-objective utility function, which integrates such factors, namely:

1) *Interestingness:* Is the ability of a view to reveal some interesting insights about the data, which is measured using the deviation-based metric $D(V_{i,b})$ (Eq. (2)).
2) *Usability:* Is the quality of the visualization in terms of providing the analyst with an understandable uncluttered representation, which is quantified via the relative bin width metric $S(V_{i,b})$ (Eq. (3)).
3) *Accuracy:* Is the ability of the view to accurately capture the characteristics of the analyzed data, which is measured in terms of the accuracy metric $A(V_{i,b})$ (Eq. (4)).

Notice that the different factors listed above are often at odds with each other. For instance, a view that contains a large number of bins can provide high accuracy, at the expense of being cluttered and difficult to understand by an analyst. To the contrary, using a small number of bins leads to a very smooth and coarse representation of the analyzed data, which can hide its particular and interesting characteristics. To capture those conflicting factors, MuVE employs a weighted sum multi-objective utility function, which is defined as follows:

$$U(V_{i,b}) = \alpha_D \times D(V_{i,b}) + \alpha_A \times A(V_{i,b}) + \alpha_S \times S(V_{i,b}), \quad (5)$$

where $D(V_{i,b})$ is the normalized deviation of binned view $V_{i,b}$ from the overall data, $A(V_{i,b})$ is the accuracy of $V_{i,b}$, and $S(V_{i,b})$ is the usability of $V_{i,b}$.

Parameters α_D, α_A and α_S specify the weights assigned to each objective in our hybrid utility function, such that $\alpha_D + \alpha_A + \alpha_S = 1$. Those weights can be user-defined so that to reflect the user's preference between the three aspects of utility. Also, notice that all objectives are normalized in the range $[0, 1]$. Accordingly, the overall multi-objective utility function takes value in the same range (i.e., $[0, 1]$), where the goal is to maximize that overall utility. Such goal is formulated as follows:

Definition 2 (Multi-Objective View Recommendation). *Given a user-specified query Q on a database D_B, a multi-objective utility function U, and a positive integer k, find the k aggregate binned views over D_Q, which have the highest utility values.*

In summary, we posit that a view is of high utility, if it shows a unique pattern that is based on accurate data and can be visually identified and appreciated by the user. For instance, referring back to our motivating example in Section 1, while Fig. 1 shows a non-binned view (i.e., accuracy of 1.0), the deviation provided by that view is only 0.17, and its usability is ~ 0. Meanwhile, Fig. 3 shows a binned version of the same view obtained at $\alpha_A = 0.2, \alpha_D = 0.6, \alpha_S = 0.2$, which results in deviation = 0.29, usability = 0.33, and accuracy = 0.30. That increase in both deviation and usability, allowed that particular view to come first on the view recommendation list (i.e., *top-1*), and enabled for an insightful visualization.

3.1 View Processing Cost

Recall that in the absence of numerical dimensions, the number of candidate views N to be constructed is equal to $N = 2 \times |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$. 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. In addition, a total of $|\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$ distance computations are needed to calculate the deviation between each pair of target and 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: $2 \times |\mathbb{M}| \times |\mathbb{F}| \times B_j$, 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

$$N_B = \sum_{j=1}^{|\mathbb{A}|} 2 \times |\mathbb{M}| \times |\mathbb{F}| \times B_j. \quad (6)$$

Furthermore, for each pair of target and comparison binned views, the three metrics/objectives listed above are to be evaluated. Evaluating those metrics incurs the following processing costs:

- *Query Execution Time*: 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 $C_c(V_{i,b})$.
- *Deviation Computation Time*: 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 $dist$.
- *Accuracy Evaluation Time*: the time required to measure the accuracy of the binned target view in representing the underlying data distribution and is denoted as $C_a(V_{i,b})$.

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

$$C(V_i) = \sum_{b=1}^{B} C_t(V_{i,b}) + C_c(V_{i,b}) + C_d(V_{i,b}) + C_a(V_{i,b}). \quad (7)$$

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

$$C = \sum_{i=1}^{N_B} C(V_i). \quad (8)$$

3.2 Search Strategy Overview

In Sections 4 and 5, we present search strategies for finding the top-k binned views for recommendation. For clarity of presentation, we break down a search strategy into two integral components, namely: 1) *Horizontal* Search, and 2) *Vertical* Search, as shown in Fig. 4. At a high level, the objective of horizontal search is to find the optimal binning for a given non-binned view, whereas the objective of vertical search is to find the top-k binned views with the highest utility values. In Section 4, we present different strategies for horizontal search, whereas in Section 5, we expand on those strategies to enable and integrate vertical search.

4 Search Schemes: Horizontal Search

The total number of binned target and comparison views are N_B as defined in Eq. (6). Evaluating the utility of each pair of those target+comparison binned views requires a total processing time $C(V_i)$, which includes the times needed for query execution, deviation computation, and accuracy evaluation. The large number of possible binned views, together with the complexity of evaluating the utility function, makes the problem of finding the optimal binning for a certain view V_i highly challenging. In the following, we present the optimal baseline scheme namely Linear Search together with our proposed schemes; i) Multi-Objective View Recommendation for Data Exploration (MuVE), ii) Upper bound based MuVE (uMuVE), and iii) Memory-aware uMuVE (MuMuVE).

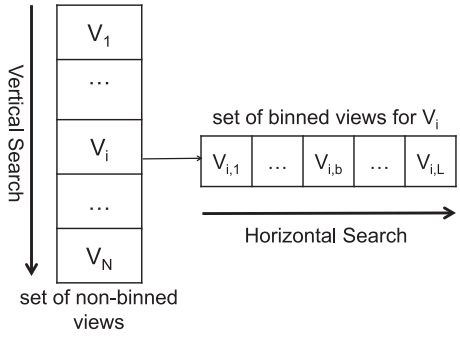


Fig. 4. Horizontal and vertical searches for recommending top-k visualizations.

4.1 Linear Search

Linear search is basically an exhaustive brute force strategy, which serves as a baseline for our evaluation. In this strategy, given a certain candidate non-binned view V_i, all its corresponding binned views are generated and the overall utility of each of those views is evaluated. Particularly, a non-binned view $V_i = (A, M, F)$ is expanded into a set of binned views: $\mathbb{V}_i = \{V_{i,1}, \ldots, V_{i,b}, \ldots, V_{i,L}\}$, where b is the number of bins, and L is the range of the continuous numerical dimension A. Consequently, the value of b that results in the highest utility is selected as the binning option for view V_i resulting in the binned view $V_{i,opt}$.

4.2 The MuVE Scheme

Similar to the linear search described above, for a given non-binned view $V_i = (A, M, F)$, our MuVE scheme considers the set of all its possible binned views: $\mathbb{V}_i = \{V_{i,1}, V_{i,2}, \ldots, V_{i,b}, \ldots, V_{i,L}\}$. Unlike linear search, however, MuVE reduces the computational costs incurred in processing that set by means of: 1) pruning unnecessary views, and 2) pruning unnecessary utility evaluations.

To easily understand MuVE, notice that our problem of searching the space and ranking binned views according to our multi-objective utility function Eq. (5) is similar to *Top-K* preference query processing. Particularly, for a view $V_{i,b}$, the three objectives $D(V_{i,b}), A(V_{i,b}), S(V_{i,b})$ can be perceived as the preference query over 3-dimensions. However, efficient algorithms for preference query processing (e.g., [14], [24]), are not directly applicable to our problem because: 1) For any view $V_{i,b}$ the values of $D(V_{i,b})$ and $A(V_{i,b})$ are not physically stored and they are computed on demand based on the binning choice b, and 2) The size of the view search space $\mathbb{V}_i$ is prohibitively large and potentially infinite. To address these limitations, MuVE adapts and extends algorithms for Top-K query processing towards efficiently and effectively solving the multi-objective view recommendation problem.

Before describing MuVE in details, we first outline a baseline solution based on simple extensions to the

Threshold Algorithm (TA) [14]. Conceptually, to adapt the well-know TA to the view recommendation model, each possible binned view $V_{i,b}$ is considered as an object with three partial scores: 1) deviation $\alpha_D D(V_{i,b})$, 2) Accuracy $\alpha_A A(V_{i,b})$, and 3) Usability $\alpha_S S(V_{i,b})$. Those partial scores are maintained in three separate lists: 1) D-list, 2) A-list, and 3) S-list, which are sorted in descending order of each score. Under the classical TA algorithm, the three lists are traversed sequentially in a round-robin fashion. While traversing, the binned view with the maximum utility seen so far is maintained along with its utility. An upper bound on the utility (i.e., threshold) is computed by applying the utility function to the partial components of the last seen view in the three different lists. TA terminates when the maximum utility seen so far is above that threshold or when the lists are traversed to completeness.

Clearly, such straightforward conceptual implementation of TA is infeasible to our problem due to the limitations mentioned before. However, recall that the usability objective S is based on the number of bins in a view and is calculated as $S(V_{i,b}) = \frac{w}{L} = \frac{1}{b}$. Hence, out of the three lists mentioned above, a sorted list S can easily be generated at a negligible processing cost. In particular, given a view V_i over a numerical dimension A of range L, MuVE progressively populates the S-list with the values $\alpha_S S(V_{i,1}), \alpha_S S(V_{i,2}), \ldots, \alpha_S S(V_{i,L})$, which are the values of the usability objective sorted in decreasing order.

One possible approach for populating the D-list and A-list is to first generate the S-list and then compute the corresponding $D(V_{i,b})$ and $A(V_{i,b})$ values for each view $V_{i,b}$. Those values are then sorted in descending order and the TA algorithm is directly applied on all three lists. Clearly, that approach has the major drawback of incurring the cost for computing the deviation and accuracy of all the possible binned views. Instead, we leverage the particular *Sorted-Random (SR)* model of the Top-K problem to minimize the number of those expensive estimation probes.

The SR model is particularly useful in the context of web-accessible external databases, in which one or more of the lists involved in an objective function can only be accessed in random and at a high-cost [14], [16], [24]. Hence, in that model, the sorted list basically provides an initial set of candidates, whereas random lists (i.e., R) are probed on demand to get the remaining partial values of the objective function. In our model, the S-list already provides that sorted sequential access, whereas the D-list and A-list are clearly external lists that are accessed at the expensive costs of computing the deviation and accuracy. Under that setting, while the S-list is generated incrementally, two values are maintained (as in [14], [24]): 1) U_{seen}: the maximum utility seen among all binned views generated so far, and 2) U_{max}: a threshold on the maximum possible utility for the binned views yet to be generated. These two values enable efficient navigation of the search space by pruning a significant number of possible binned views as well as utility evaluations, which is achieved using two simple techniques:

Incremental Evaluation: The main idea is to calculate the different components of the utility function $U(V_{i,b})$ incrementally and terminate the calculation once it is clear that $V_{i,b}$ is not the optimal binned view. To achieve this, when a candidate binned view $V_{i,b}$ is considered, its $S(V_{i,b})$ value is compared to the maximum utility seen so far, (i.e., U_{seen}), then the calculation of its $D(V_{i,b})$ and $A(V_{i,b})$ values are eliminated (i.e., pruned) if $\alpha_D + \alpha_A + \alpha_S S(V_{i,b}) \leq U_{seen}$. The idea is that since each of $D(V_{i,b})$ and $A(V_{i,b})$ is bounded to 1.0, then a binned view $V_{i,b}$ that satisfies this condition will never have a utility greater than U_{seen}, which makes evaluating its deviation and accuracy unnecessary. Such view will incur no processing costs since $S(V_{i,b})$ is readily available given b, whereas the calculations of $D(V_{i,b})$ and $A(V_{i,b})$ are pruned.

If the above condition is not satisfied, instead of calculating both $D(V_{i,b})$ and $A(V_{i,b})$, further incremental evaluation is performed. Particularly, MuVE decides an order of evaluation of those two objectives. If $D(V_{i,b})$ is evaluated first, then if $\alpha_D D(V_{i,b}) + \alpha_A + \alpha_S S(V_{i,b}) \leq U_{seen}$, then $V_{i,b}$ is safely pruned without the need for evaluating its accuracy. Alternatively, if $A(V_{i,b})$ is evaluated first, then if $\alpha_D + \alpha_A A(V_{i,b}) + \alpha_S S(V_{i,b}) \leq U_{seen}$, then the deviation objective is not calculated and $V_{i,b}$ is pruned. The evaluation order of these two objectives is very important for pruning of low utility views. To decide the evaluation oder of those two objectives, MuVE applies a simple priority function, such that if

$$\frac{\alpha_A}{C_t(V_{i,b}) + C_a(V_{i,b})} > \frac{\alpha_D}{C_t(V_{i,b}) + C_c(V_{i,b}) + C_d(V_{i,b})}, \quad (9)$$

then $A(V_{i,b})$ is evaluated first, otherwise $D(V_{i,b})$ is the one to be evaluated first. The idea is to give higher priority to evaluating an objective if it incurs less processing cost and/or contributes more to the utility function that is to be maximized. Recall that $C_t(V_{i,b})$, $C_c(V_{i,b})$, $C_d(V_{i,b})$, $C_a(V_{i,b})$ are the costs of evaluating the target view, comparison view, deviation, and accuracy, respectively. To estimate such costs for a binned view $V_{i,b}$, MuVE simply maintains a moving average of each of those costs over the previous $V_{i,1}, V_{i,2}, \ldots, V_{i,b-1}$ binned views. Particularly, whenever a short circuit fails and an objective is evaluated, the cost for evaluating the operations involved in that objective is updated as: $C_x(V_{i,b}) = \beta C_x(V_{i,b-1}) + \frac{1-\beta}{b-2} \sum_{j=1}^{b-2} C_x(V_{i,j})$, where x is any of the four operations listed above, and $\beta = 0.825$ to give more weight to the most recent costs. In our experiments (Section 7), we consider other options for setting the priority function and discuss the trade-offs between those options.

Early Termination: when a binned view $V_{i,b}$ is considered for evaluation, the threshold U_{max} is updated to $U_{max} = \alpha_D + \alpha_A + \alpha_S S(V_{i,b})$. That is, assuming that $V_{i,b}$ will receive the maximum score of 1.0 under both the deviation and accuracy objectives. In that case, if $U_{seen} \geq U_{max}$, then it is guaranteed that the actual utility of $V_{i,b}$ cannot exceed U_{seen}. Moreover, since all the following views starting at $V_{i,b+1}$ will have lower S values, they are also guaranteed to provide utilities less than U_{seen}. Hence, those views are pruned and early termination is reached.

Example 2. Consider applying MuVE search on a non-binned view V_i, which is represented by 5 binned views $V_{i,2}$ to $V_{i,6}$ (Fig. 5). Further, consider that $k = 1$, $\alpha_D = 0.2$, $\alpha_A = 0.6$ and $\alpha_S = 0.2$. For each binned view $V_{i,b}$, its usability $S(V_{i,b})$ is already known and the binned views are ordered accordingly. However, the values of the other two objectives $A(V_{i,b})$ and $D(V_{i,b})$ are calculated as MuVE progresses. $U(V_{i,b})$ is the utility of the binned view $V_{i,b}$ if all three objectives are known, otherwise, it specifies an upper bound on that utility. U_{seen} is the maximum utility seen so far and U_{max} is the threshold on the maximum possible utility. Fig. 5 shows a snapshot of how MuVE returns the top-1 binned view for V_i. In the first iteration (Fig. 5a), U_{max} is set as $\alpha_D + \alpha_A + \alpha_S S(V_{i,2}) = 0.90$, then

(a) First Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{seen} = 0.62$
$A(V_{i,b})$	0.60					$U_{max} = 0.90$
$D(V_{i,b})$	0.80					
$U(V_{i,b})$	0.62	$\leq$0.87	$\leq$0.85	$\leq$0.84	$\leq$0.83	

(b) Second Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{seen} = 0.62$
$A(V_{i,b})$	0.60	**0.65**				$U_{max} = 0.87$
$D(V_{i,b})$	0.80					
$U(V_{i,b})$	0.62	$\leq$0.66	$\leq$0.85	$\leq$0.84	$\leq$0.83	

(c) Third Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{seen} = 0.62$
$A(V_{i,b})$	0.60	0.65	0.60			$U_{max} = 0.85$
$D(V_{i,b})$	0.80	0.56	Pruned			
$U(V_{i,b})$	0.62	0.57	$\leq$0.61	$\leq$0.84	$\leq$0.83	

(d) After the Final Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{seen} = 0.74$
$A(V_{i,b})$	0.60	0.65	0.60	0.80	1.00	$U_{max} = 0.83$
$D(V_{i,b})$	0.80	0.56	Pruned	0.55	0.52	
$U(V_{i,b})$	0.62	0.57	$\leq$0.61	0.63	0.74	

Fig. 5. Example: The MuVE scheme.

(a) First Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{max} = 0.90$
$A(V_{i,b})$	0.60					
$D(V_{i,b})$						
$U(V_{i,b})$	$\leq$0.66	$\leq$0.87	$\leq$0.85	$\leq$0.84	$\leq$0.83	

(b) Second Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{max} = 0.87$
$A(V_{i,b})$	0.60	0.65				
$D(V_{i,b})$						
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	$\leq$0.85	$\leq$0.84	$\leq$0.83	

(c) Fifth Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{max} = 0.83$
$A(V_{i,b})$	0.60	0.65	0.60	0.80	1.00	
$D(V_{i,b})$						
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	$\leq$0.61	$\leq$0.72	$\leq$0.83	

(d) After the Final Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$U_{max} = 0.83$
$A(V_{i,b})$	0.60	0.65	0.60	0.80	1.00	
$D(V_{i,b})$	Pruned				0.52	
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	$\leq$0.61	$\leq$0.72	0.74	

Fig. 6. Example: The uMuVE scheme.

both the accuracy and deviation of the first binned view $V_{i,2}$ are computed (shown as a highlighted column in Fig. 5a). Since $V_{i,b}$ is the only seen view, therefore, $U_{seen} = U(V_{i,2}) = 0.62$. In the second iteration (Fig. 5b), MuVE considers the next view (i.e., $V_{i,3}$). The new value of U_{max} is 0.87 calculated by $\alpha_D + \alpha_A + \alpha_S S(V_{i,3})$. According to our priority function, $V_{i,3}$ is probed for accuracy and its new upper bound on utility is computed as $\alpha_D + \alpha_A$ $A(V_{i,3}) + \alpha_S S(V_{i,3}) = 0.66$, which is stored in $U(V_{i,3})$. Since that value of 0.66 is greater than the current value of U_{seen}, then $V_{i,3}$ can possibly have a utility greater than U_{seen}, so it is further probed for deviation. In the third iteration (Fig. 5c), $V_{i,4}$ is probed for accuracy. Its updated upper bound on utility $U(V_4) = 0.61$, which is less than U_{seen}, therefore its deviation computation is pruned. Similar to $V_{i,2}$, the next two views $V_{i,5}$ and $V_{i,6}$ are also fully probed (both accuracy and deviation are evaluated). Finally, MuVE selects $V_{i,6}$ as the top-1 view, which has the highest utility.

In Example 2, MuVE managed to prune one deviation calculation for $V_{i,4}$. In general, the amount of pruning achieved by MuVE depends on several factors including the data distribution and the weights of each objective. To attain even higher pruning power, our uMuVE and MuMuVE schemes are proposed next.

4.3 The uMuVE Scheme: Upper Bound Based MuVE

Similar to the TA algorithm, MuVE visits the sequence of possible binned views one at a time. For each visited view $V_{i,b}$, the objective values $A(V_{i,b})$ and $D(V_{i,b})$ are either immediately calculated or pruned. Hence, all the necessary processing is completed once a view is visited and no backtracking is needed. Alternatively, in this section, we propose uMuVE, which extends the *Upper Algorithm* [8], [24] and allows for interleaved processing of views. Particularly, uMuVE follows a greedy approach, in which it evaluates the most promising views first so that to minimize the total processing time while providing the same results as our original MuVE.

Like MuVE, uMuVE also maintains U_{max}, a threshold on the highest possible utility of a view in the S-list, which has not yet been visited. Additionally, each binned view $V_{i,b}$ has an *upper bound* $U_{upper}(V_{i,b})$ on its utility, which is the maximum possible utility a view $V_{i,b}$ can have. In uMuVE a *priority queue* PQ is maintained based on the upper bound scores of the views. When a view is retrieved from the S-list, its $U_{upper}(V_{i,b})$ equals to $\alpha_D + \alpha_A + \alpha_S S(V_{i,b})$ and it is added to PQ. Once any or both of the objectives (i.e., deviation or accuracy) is calculated, $U_{upper}(V_{i,b})$ of the view is updated accordingly.

For example, in Example 2, the first view retrieved from the S-list and added to PQ is $V_{i,2}$ (Fig. 6a). According to our earlier explanation, initially $U_{upper}(V_{i,2}) = U_{max} = \alpha_D + \alpha_A + \alpha_S S(V_{i,2}) = 0.90$. In the first iteration, according to our priority function (Eq. (9)), accuracy of $V_{i,2}$ is computed and its upper bound on utility is updated to 0.66.

In each iteration, uMuVE decides to process one out of two views: 1) view $V_{i,b+1}$, which is the next view in the S-list, or 2) V_h, which is the view with the highest U_{upper} in PQ. To decide between those two views, the upper bound of V_h is compared against U_{max} and the following two cases are considered:

Case 1: $U_{upper}(V_h) \leq U_{max}$*:* If the upper bound of V_h is lower than the upper bound of the unseen views and there are still views that are not added to PQ yet. Then, this is an indication that the next view in S-list can have a higher upper bound on its utility as compared to all of the views in PQ. Particularly, U_{upper} of the next binned view $V_{i,b+1}$ is computed and U_{max} is updated as $U_{max} = U_{upper}(V_{i,b+1})$.

To further illustrate that case, consider Fig. 6 again. As $U_{upper}(V_{i,2})$ is less than U_{max}, the decision on pruning or probing of $V_{i,2}$ is deferred for now (shown as dark highlighted column) and it will be kept in memory until uMuVE comes back to it. In the second iteration (Fig. 6b), U_{max} is updated according to the new view $V_{i,3}$ and $V_{i,3}$ is added to PQ. After probing $V_{i,3}$ on accuracy, its upper bound on utility becomes less than U_{max} and uMuVE retrieves the next view from S-list. In the next three iterations, uMuVE will add the next three views $V_{i,4}$, $V_{i,5}$, $V_{i,6}$ in PQ and probe them for accuracy only, as shown in Fig. 6c.

Case 2: $U_{upper}(V_h) > U_{max}$ *or S-list is empty:* If $U_{upper}(V_h)$ is greater than U_{max}, then this is an indication that V_h can be the top-1 view. If V_h is already fully probed then it is

returned as the top-1 view and the rest of the views are removed from PQ. However, if it is partially probed then its other objective is calculated and $U_{upper}(V_h)$ is updated accordingly.

If S-list is empty, uMuVE selects the view with highest upper bound V_h, probe it further and update its $U_{upper}(V_h)$. To illustrate this case consider Fig. 6d, in which uMuVE has added all possible views to the PQ and S-list is empty. Now, uMuVE will select a view from the front of PQ i.e., $V_{i,6}$ and compute its deviation objective. The final utility of $V_{i,6}$ is 0.74 and the upper bound of every other view is less than this utility, therefore, $V_{i,6}$ is returned as top-1 view. Fig. 6 clearly shows that uMuVE is able to prune three more deviation evaluations as compared to MuVE.

4.4 The MuMuVE Scheme: Memory-Aware uMuVE

Our machines have limited computational and memory resources. Therefore, along with the processing cost of proposed search schemes, it is also important to understand their memory requirements. For instance, consider our proposed schemes MuVE and uMuVE, MuVE's memory usage is negligible since it considers only one view at a time. However, uMuVE stores multiple views in the priority queue, therefore, it uses more memory and its memory requirements need to be quantified and optimized. In the classical Upper scheme [8], [24], objects are points in multidimensional space and partial probes only require to store single value per dimension. Therefore, memory is not a critical problem in that scenario. However, in our case, each object is a view and view data is stored in memory. This brings forth the addition challenge of optimizing memory requirements of uMuVE. In this section, we address that challenge by quantifying the memory needs of uMuVE and proposing the memory-aware uMuVE scheme (MuMuVE).

As explained in Section 4.3, the decision on the views in the priority queue remain pending, unless there is an indication that those views should be probed further or pruned. The memory requirements for uMuVE depend on the order in which views are visited. For instance, for a non-binned view V_i, if the top-1 binned view is seen earlier in the search, memory needs will be minimal. In the worst case scenario, the data for every binned view $V_{i,b}$ is stored in memory. The amount of memory used by a binned view $V_{i,b}$ denoted as $M_U(V_{i,b})$, equals to the number of bins b of $V_{i,b}$. Then, a binned view $V_{i,b}$ in PQ, can be in one of the following three states:

1) $A(V_{i,b})$ & $D(V_{i,b})$ are known: That is, view $V_{i,b}$ has been fully probed and both objectives have been evaluated. Therefore, the data for comparison and target view is not required anymore. Hence, $M_U(V_{i,b}) = 0$.
2) $A(V_{i,b})$ is known & $D(V_{i,b})$ is unknown: That is, the accuracy of view $V_{i,b}$ has been computed. Specifically, the underlying target view of $V_{i,b}$ has been retrieved and it will be kept in memory until $V_{i,b}$ is pruned or probed for deviation. Therefore, $M_U(V_{i,b}) = b$.
3) $D(V_{i,b})$ is known & $A(V_{i,b})$ is unknown: That is, the deviation of view $V_{i,b}$ has been evaluated. Particularly, the underlying target and comparison views of $V_{i,b}$ were retrieved. After computing deviation, the comparison view is discarded, but the target view will remain in main memory because it may be needed if uMuVE decides to evaluate the accuracy of this view. Therefore, $M_U(V_{i,b}) = b$.

Let L_i be the range of the dimension attribute of the non-binned view V_i, then the maximum memory needed to search for the top-1 binned view is

$$M_U(V_i) = \sum_{b=2}^{L_i} M_U(V_{i,b}) = \sum_{b=2}^{L_i} b.$$

In Example 2, the amount of memory used by uMuVE (Fig. 6) is $M_U(V_i) = \sum_{b=2}^{6} b = 2 + 3 + 4 + 5 + 6 = 20$. This is because uMuVE had all 5 views in the memory.

Clearly, uMuVE always has lower processing cost as compared to MuVE, but it has additional memory needs. To balance the trade-off between memory and processing time, a memory-aware version of uMuVE (MuMuVE) is proposed next.

MuMuVE is an extension of uMuVE for *memory-bounded* evaluation of views. The new scheme attempts to minimize view probes for a given amount of memory M_L. The main idea is when the memory utilization gets close to M_L, instead of adding more views to PQ, give preference to probing those views which are already in PQ. Consequently, memory will be *evicted*.

As explained previously, uMuVE selects a view V_h with the highest U_{upper} from PQ and if $U_{upper}(V_h) < U_{max}$, moves a binned view V_S from S-list to PQ. However, for MuMuVE before adding V_S to PQ, we also need to check if there is enough space available. Therefore, the condition is modified as: if $U_{upper}(V_h) < U_{max}$ & $M_U(V_S) < M_L$ then add V_S to PQ. V_S is not added to PQ if any of the two conditions is violated. Hence, if $U_{upper}(V_h) < U_{max}$ is false, MuMuVE probes V_h further. If $M_U(V_S) < M_L$ is false, that means $M_U(V_i)$ has reached the bound M_L. MuMuVE will choose a partially probed view $V_{h'}$ from PQ to complete its utility evaluation. This ensure that the memory occupied by $V_{h'}$ is evicted and more views can be added to PQ. The decision about which view to evict is critical, as it affects the processing time. We consider the following three options:

1) *Max-Bins*: In this option, MuMuVE chooses the view with the maximum number of bins from the partially probed views in PQ. Although, there is no indication that this view might be the top-1 view, however, it ensures that the maximum possible memory is evicted by a single probe.

   Again, consider Example 2 and assume $M_L = 10$. MuMuVE starts adding views from S-list to PQ as in uMuVE. By the end of the third iteration (Fig. 7a), $V_{i,2}$, $V_{i,3}$ and $V_{i,4}$ are already in PQ. The amount of memory used $M_U(V_i)$ is 9. The next view in S-list $V_{i,5}$ require 5 locations while there is only 1 available, therefore, MuMuVE needs to evict memory before adding $V_{i,5}$ to PQ. According to max-bins, MuMuVE probes the view having maximum bins i.e., $V_{i,4}$ (Fig. 7b). Consequently, there is enough space to add $V_{i,5}$ as shown in the final state of the fourth iteration in Fig. 7c. After the final iteration (Fig. 7d), this scheme is able to prune two objective evaluations as compared to four of uMuVE. However, in worst case it only used 50 percent of the space compared to uMuVE.

(a) Third Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60			$M_U(V_i) = 9$
$D(V_{i,b})$						
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	$\leq$0.61	$\leq$0.84	$\leq$0.83	

(b) Fourth Iteration: Initial State

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	**0.60**			$M_U(V_i) = 5$
$D(V_{i,b})$			**0.54**			
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	**0.52**	$\leq$0.84	$\leq$0.83	

(c) Fourth Iteration: Final State

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60	0.80		$M_U(V_i) = 10$
$D(V_{i,b})$			0.54			
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	0.52	$\leq$0.72	$\leq$0.83	

(d) After the Final Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60	0.80	1.00	
$D(V_{i,b})$	Pruned		0.54	0.55	0.52	
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	0.52	0.63	0.74	

Fig. 7. Example: The MuMuVE scheme (Max-Bins).

(a) Third Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60			$M_U(V_i) = 9$
$D(V_{i,b})$						
$U(V_{i,b})$	$\leq$0.66	$\leq$0.66	$\leq$0.61	$\leq$0.84	$\leq$0.83	

(b) Fourth Iteration:Initial State

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60			$M_U(V_i) = 7$
$D(V_{i,b})$	0.80					
$U(V_{i,b})$	0.62	$\leq$0.66	$\leq$0.61	$\leq$0.84	$\leq$0.83	

(c) Fourth Iteration: Final State

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60	0.80		$M_U(V_i) = 4$
$D(V_{i,b})$	0.80	0.56				
$U(V_{i,b})$	0.62	0.57	$\leq$0.61	$\leq$0.72	$\leq$0.83	

(d) After the Final Iteration

	$V_{i,2}$	$V_{i,3}$	$V_{i,4}$	$V_{i,5}$	$V_{i,6}$	
$S(V_{i,b})$	0.50	0.33	0.25	0.20	0.17	$M_L = 10$
$A(V_{i,b})$	0.60	0.65	0.60	0.80	1.00	
$D(V_{i,b})$	0.80	0.56	Pruned	0.55	0.52	
$U(V_{i,b})$	0.62	0.57	$\leq$0.61	0.63	0.74	

Fig. 8. Example: The MuMuVE scheme (Max-Utility).

2) *Max-Utility*: In this option, MuMuVE chooses the view $V_{h'}$ having maximum upper bound U_{upper}. However, if number of bins of $V_{h'}$ is lower than the number of bins of V_S then it will keep choosing $V_{h'}$ views until there is enough space. Therefore, as result MuMuVE evaluates unnecessary objectives and that will increase processing cost as compared to MuVE. On the other side there is possibility that the view MuMuVE has fully probed might be the top-1, which will end the search.

   Consider Example 2 again and assume $M_L = 10$. Similar to max-bins, $V_{i,2}$, $V_{i,3}$, $V_{i,4}$ are added to PQ and M_L is reached (Fig. 8a). According to max-utility, the view having maximum upper bound on utility i.e., $V_{i,2}$ is probed further (Fig. 8b). However, the available space ($M_L - M_U$) is still not enough to accommodate $V_{i,5}$. Therefore, MuMuVE probe $V_{i,3}$ (Fig. 8c), which will reduce the used memory space M_U to 4. Then, $V_{i,5}$ can be added to PQ as shown in the final state of fourth iteration in Fig. 8c. Max-utility pruned one objective evaluation as compared to two of max-bins.
3) *Max-Min-Utility:* Memory requirements and processing time of the max-utility scheme can be further reduced by keeping track of lower bound on the utility of the views in PQ. Particularly, when the memory is full, view $V_{h'}$ with the maximum utility from PQ is probed further. Utility of this $V_{h'}$ becomes lower bound on the utility i.e., $U_{lower} = U(V_{h'})$ and all views $V_{i,b}$ from PQ which have $U_{upper}(V_{i,b}) < U_{lower}$ are removed, because none of these can be top-1 view.

Consider Fig. 8 again, and assume U_{lower} is also maintained. In initial state of the fourth iteration (Fig. 8b) after view $V_{i,2}$ is fully probed the lower bound on utility is updated as $U_{lower} = 0.62$. As the upper bound on the utility of $V_{i,4}$ is less than U_{lower}, which means $V_{i,4}$ can not be in top-1 therefore it is removed from PQ. Now, there is enough memory available to probe the next view. After the final step, max-min-utility is able to prune two objective evaluations compared to max-utility.

5 Search Schemes: Vertical Search

Recall that the goal of this work is to recommend the top-k visualizations that maximize our multi-objective utility function. In the previous section, we discussed horizontal search strategies, which find the optimal binned $V_{i,opt}$ for a given non-binned view V_i. As discussed earlier, the space of possible non-binned views, is of size $N = 2 \times |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$. In the case where $\mathbb{A}$ is a set of numerical dimensions, then the total number of corresponding possible binned views is N_B, where $N_B = \sum_{j=1}^{|\mathbb{A}|} 2 \times |\mathbb{M}| \times |\mathbb{F}| \times B_j$ and the goal is simply to find the top-k binned views across those N_B views. We note, however, that recommending two different binned views that correspond to the same non-binned views adds little value to the analyst and is rather redundant. Hence, if V_{x,b_1} and V_{y,b_2} are two views in the top-k list, then $x \neq y$. Consequently, we propose the following vertical search strategies.

In our first strategy for vertical search, we extend linear search (as described in the previous section) for the purpose of finding the top-k recommendations. Particularly, in this simple strategy, the set of all possible non-binned views $\mathbb{V}$ is traversed sequentially in an exhaustive manner. Then, each view $V_i \in \mathbb{V}$ is expanded and searched horizontally to find its optimal binned view $V_{i,opt}$. As linear search finishes scanning $\mathbb{V}$, the optimal binned view corresponding to each view V_i is identified, and out of those, the k with the highest utility are the ones to be recommended.

Note, however, that under this vertical linear search, the vertical and horizontal searches are clearly decoupled. Hence, while the vertical search is performed linearly, the choice of the horizontal search strategy is open. Given the algorithms discussed so far, this allows for the combinations denoted as follows: *linear-linear:* in which linear search is used for both the vertical and horizontal searches, and *MuVE-Linear:* in which linear search is used for vertical search, whereas the optimized MuVE, as described in the previous section, is used for horizontal search. Obviously, the latter combination allows leveraging the optimizations offered by MuVE to reduce the cost of each horizontal search. Towards further

optimizations, in the following we discuss extending MuVE, uMuVE and MuMuVE to perform both the vertical and horizontal searches (i.e., MuVE-MuVE, uMuVE-Linear, uMuVE-uMuVE, MuMuVE-Linear, MuMuVE-MuMuVE).

Extending MuVE to perform both horizontal and vertical searches is straightforward. To explain that extension, recall that for performing horizontal search on a non-binned view V_i over a numerical dimension of range L, MuVE progressively populates the S-list with the values $\alpha_S S(V_{i,1})$, $\alpha_S S(V_{i,2}), \ldots, \alpha_S S(V_{i,L})$, which are the values of the usability objective sorted in decreasing order. Hence, to allow vertical search, MuVE traverses the set of non-binned views $\mathbb{V}$ in a round-robin fashion, where in a round r each view $V_i \in \mathbb{V}$ is appended to the S-list as $V_{i,r}$, given that r is less than the maximum number of bins that is possible for that view. Adding a view $V_{i,r}$ to the S-list triggers evaluating the multi-objective utility function $U(V_{i,r})$. That evaluation is performed similar to the one described above for the horizontal search, except that the pruning conditions employed for the incremental evaluation are set for top-k instead of top-1. Evaluating new views continues until all possible binned views are generated or until early termination is reached, then the top-k views with the highest utility are returned to the user.

In comparison to MuVE-Linear described above, using MuVE for both vertical and horizontal search clearly offers further reductions in cost by means of increasing the number of pruned operations. To explain this, consider an uninteresting view V_i (i.e., a view with low deviation). If that view is considered in isolation, as in MuVE-Linear, then significant processing time will be spent on finding $V_{i,opt}$. However, $U(V_{i,opt})$ is expected to still be very small compared to the other views, which are more interesting. Under MuVE, however, those interesting views will lead to increasing the value of U_{seen}, which in turn allows for pruning many of the objective evaluations that were to be performed on V_i.

Extending uMuVE to perform vertical and horizontal search is similar to MuVE. Like MuVE, uMuVE traverses the set of non-binned views $\mathbb{V}$ in round robin fashion and in each round r it appends each $V_{i,r}$ to S-list. However, uMuVE also maintains a priority queue PQ and moves views from S-list to PQ when none of the views in PQ are promising enough to be probed further. When the utility of a fully probed view V_h becomes greater than U_{max} it is recommended as top-k and removed from PQ. Further, all of the views with same A, M and F as V_h are also removed from PQ to ensure that the recommended views are diverse.

Progressive Results: A key advantage of using uMuVE in both horizontal and vertical direction is that it can produce *progressive results*, i.e., results as they become available rather than waiting for complete evaluation of all of the views. Specifically, after returning the top-1 view, if more views are required uMuVE continues to probe more views and complete their utility evaluations. Then, uMuVE returns the next view with the highest utility and it keeps going until all of the top-k views are returned. This feature can be particularly beneficial in an interactive data exploration scenario, where as soon as the first few results are known, the user may decide to terminate the current search and begin a next search by changing some of the input parameters.

Additional Aggregate Queries: We note that while in this paper we consider only aggregation with a single numeric dimension (i.e., single group-by attribute), our techniques are directly applicable to the more general scenario, in which there are multiple numeric group-by attributes. Such aggregations will result in multi-column views that can be visualized as multi-dimensional or stacked bar charts. Hence, more memory is required to store those multi-column views but the process of the objective computation would remain the same.

Another interesting setting to consider is when some of the possible aggregations have a numerical dimension attribute, whereas others are based on a categorical dimension attribute. In that case, recommending the top-k interesting views is rather challenging as it requires the employed utility function to be able to fairly compare the utility provided by those two different kinds of views. One idea is to adapt our multi-objective function (Eq. (5)) such that if a view V_i is based on a categorical dimension, then its overall utility is computed as follows: 1) the deviation of V_i is computed as in Eq. (2), 2) the accuracy of V_i is always equal to 1.0 since no summarization is performed, and 3) the usability of V_i is also equal to 1.0 since categorical values cannot be aggregated in larger bins. However, that simple adaptation is expected to always assign high utility values to those views based on categorical dimensions because their accuracy and usability will always receive a perfect score of 1.0. This is clearly in contrast with our goal of ensuring a fair comparison between different kinds of views. Hence, our utility function needs to incorporate additional measures to enforce that fairness (e.g., diversification [21], [22]). A detailed exploration of that problem is part of our future work.

6 Experimental Testbed

We perform extensive experimental evaluation to measure the efficiency of the different 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 [31] to support numerical dimensional values, binned aggregation, and the different search strategies presented in this paper. Our experiments are performed on a Corei7 machine with 16GB of RAM memory. The platform is implemented in Java, and PostgreSQL is used as the backend database management system.

Schemes: We investigate the performance of the different combinations of the vertical and horizontal search strategies presented 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 vertical search. For instance, in *MuVE-Linear*, MuVE is used for horizontal search, whereas linear search is applied for vertical search.

Data Analsyis: We assume a data exploration setting in which multi-dimensional datasets are analyzed. We use three datasets: *DIAB:* dataset of diabetic patients [1], *NBA:* dataset of basketball players [4] and *CENSUS:* dataset of adult census income [2]. The independent numeric attributes of each dataset are used as dimensions, whereas the observation attributes are used as measures. For instance, in the DIAB dataset, dimensions are selected from age, BMI, etc., whereas measures are selected from insulin level, glucose concentration, etc.

The DIAB, NBA and CENSUS datasets have 9, 28 and 15 attributes, respectively. In our default setting, we select 3 dimensions, 3 measures, and 3 aggregate functions. Table 3 shows the range of each dimension A for every dataset and

TABLE 3
Details of Datasets

Datasets	Number of tuples	Range of A_1	Range of A_2	Range of A_3	Number of Views
DIAB	768	21-81	0-67	0-199	2,961
NBA	651	19-38	1-83	1-2981	27,765
CENSUS	32,561	17-90	1-16	1-99	1,701

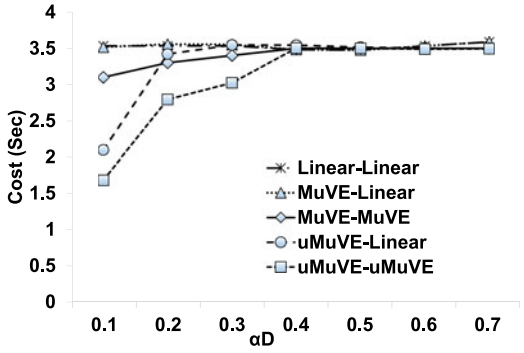


Fig. 9. DIAB: Impact of α_A and α_D on cost, when $\alpha_S = 0.2$.

accordingly the number of possible views are also shown. 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, $\alpha_D = 0.2$, $\alpha_A = 0.2$, $\alpha_S = 0.6$ and $k = 5$, unless specified otherwise. The input queries for each dataset are: DIAB: `SELECT * FROM DIAB WHERE Pregnancies>3`, NBA: `SELECT * FROM NBA WHERE team=GSW`, and CENSUS: `SELECT * FROM CENSUS WHERE income>50K.`

Performance: We evaluate the efficiency of the different recommendations strategies in terms of (1) *Cost:* As mentioned in Section 3, the cost of a strategy is the total cost incurred in processing all the candidate binned views, which is measured in wall clock time, and (2) *Fully Probed Views:* Count of the views $V_{i,b}$ for which both objectives $D(V_{i,b})$ and $A(V_{i,b})$ were calculated. Each performance metric is reported based on the average of 10 different executions.

7 Experimental Evaluation

In the following experiments, we evaluate the performance of our techniques under different parameter settings.

Impact of the α parameters (Figs. 9, 10 and 11): In this set of experiments, we measure the impact of the α values on processing time (i.e., cost). Figs. 9, 10 and 11 show how the cost of the different schemes is affected by changing the values of α_D, α_A and α_S.

In Figs. 9 and 10, α_S is set to constant 0.2 while α_A and α_D are changing. In particular, as shown in the figures, α_D is increased, while α_A is implicitly decreased and is easily computed as $\alpha_A = 1 - \alpha_D + \alpha_S$. Fig. 9 shows that *Linear-Linear* has almost same cost for all values of α_S, which is expected since it performs exhaustive search over all combinations of A, M, F, and B. Therefore, its cost depends on the number of all possible combinations, irrespective of the values of α.

Fig. 9 also shows that *MuVE-MuVE* has lower cost than *Linear-Linear* and *MuVE-Linear*, especially in the region where α_D is low and correspondingly, α_A is high. This is because interesting views with high accuracy will lead to a higher U_{seen}, which in turn allows for pruning the less interesting views during the vertical search. Furthermore, Fig. 9 also shows that *uMuVE-uMuVE* offers the lowest cost, especially when $\alpha_D < 0.4$. For instance, at $\alpha_D = 0.2$, *uMuVE-uMuVE* has almost 30 percent lower cost as compared to *MuVE-MuVE*. This is because after evaluating one objective of a view, *uMuVE* delays probing that view in full until the partial utility of that view becomes higher than the current U_{max}. Hence, it prunes many unnecessary deviation evaluations. The reduction in cost of the *MuVE* variants can be further understood using Fig. 10, in which we plot the number of views that are probed in full (i.e., both deviation and accuracy are evaluated). Fig. 10 shows that *MuVE-MuVE* and *uMuVE-uMuVE* fully probe a very low number of views at the high values of α_D. Interestingly, however, that large reduction in the number of probed views does not translate into cost saving as it has been the case at high α_A (Fig. 9). This is because at high α_D, *MuVE-MuVE* and *uMuVE-uMuVE* mainly prune the operations for computing accuracy, whereas at high α_A mainly the operations for computing deviation are pruned, which typically incur higher processing cost than that needed for computing accuracy.

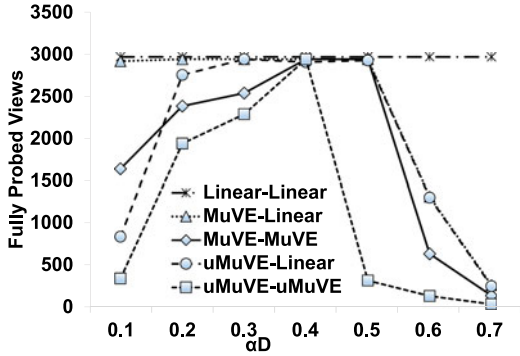


Fig. 10. DIAB: Impact of α_A and α_D on the number of fully probed views, when $\alpha_S = 0.2$.

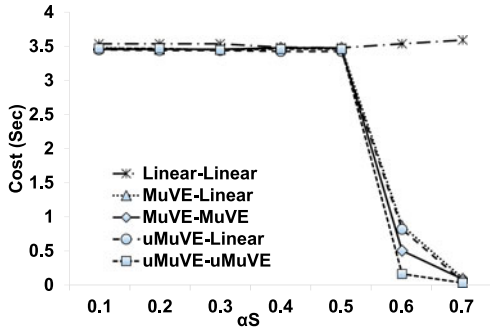
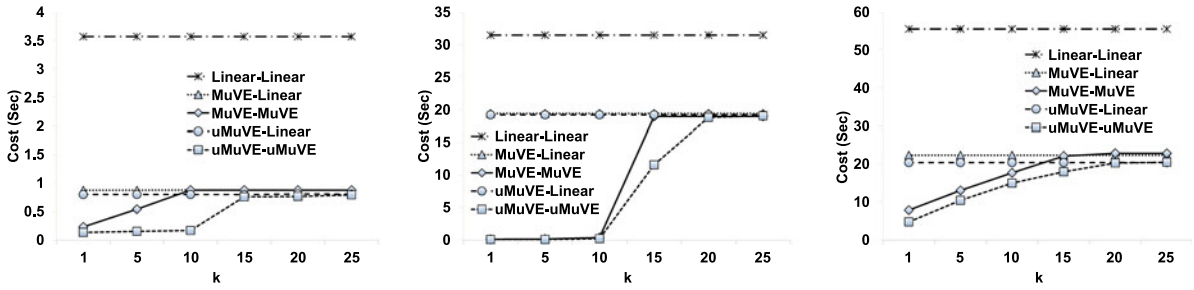


Fig. 11. DIAB: Impact of α_D and α_S on cost, when $\alpha_A = 0.2$.

In Fig. 11, $\alpha_A = 0.2$, whereas α_S is increasing and accordingly α_D is decreasing. Fig. 11 shows the effect of changing α_S and α_D values on cost. Particularly, Figs. 11 shows that the MuVE schemes have almost same cost as *Linear-Linear* for smaller values of α_S, but outperform it as the value of α_S increases. For instance, in Fig. 11 at $\alpha_S > 0.5$, all four schemes show more than 70 percent reduction in cost as compared to *Linear-Linear*. This happens because in the *MuVE* and *uMuVE* schemes, when α_S is high, there are more chances of applying the short circuiting and early termination conditions based on the usability value, and in turn pruning many of the operations required for evaluating deviation and accuracy. The amount of achieved



(a) DIAB (b) NBA (c) CENSUS

Fig. 12. Impact of k on cost.

pruning is further increased under *MuVE-MuVE* and *uMuVE-uMuVE*, which is able to prune those operations during both the vertical and horizontal searches. For instance, Fig. 11 shows that *uMuVE-uMuVE* reduces the processing cost by almost 75 percent, compared to *uMuVE-Linear*, at $\alpha_S = 0.6$.

Impact of k (Fig. 12): In the previous experiments, the value of k is set to 5 (i.e., top-5 views are recommended). Fig. 12 shows that *Linear-Linear*, *MuVE-Linear* and *uMuVE-Linear* are all insensitive to the increase in the value of k. This is because *Linear-Linear* is exhaustive search, whereas *MuVE-Linear* and *uMuVE-Linear* also performs an exhaustive vertical search. For instance, in Fig. 12a, in case of top-1 *MuVE-MuVE* reduces the cost by up to 90 percent compared to the *Linear-Linear* scheme, while the reduction offered by *uMuVE-uMuVE* is up to 85 percent compared to the *MuVE-MuVE* scheme.

Priority Function Analysis (Fig. 13): As mentioned in Section 4.2, we consider different options for setting our priority function for ordering the evaluation of objectives. The options that we considered are: 1) *Random:* Randomly chooses the objective to evaluate first, 2) *Deviation-First:* Always computes the deviation objective first, 3) *Accuracy-First:* Always computes the accuracy objective first, 4) *Weights-Based:* Selects the objective which has more weight as it contributes more to the objective function, and 5) *Hybrid:* Selects an objective based on its weight and evaluation cost (as in Eq. (9)).

Fig. 13 shows the cost of *MuVE-MuVE* in terms of the number of fully probed views when incorporating each of the options listed above. In Fig. 13, $\alpha_S = 0.2$, whereas α_D is increasing and accordingly α_A is decreasing.

Fig. 13 shows that for low value of α_D, number of fully probed views for accuracy-first scheme are lower than deviation-first because of the short circuiting of deviation objective evaluation. However, for high values of α_D, number of fully probed views for deviation-first are lower because of the short circuiting of accuracy objective evaluation. Hybrid scheme captures the advantage of both deviation-first and accuracy-first schemes.

Scalability (Fig. 14): From Section 3.1, the theoretical complexity of our recommendation problem is linear in terms of the number of dimensions A, expressed as cA, where c is the product of number of measures, aggregate functions and bin settings. While such complexity applies to both *Linear*, *MuVE* and *uMuVE*, in practice, however, c is much smaller for *MuVE* and *uMuVE* due to pruning. For example, Fig. 14 shows our results on the NBA data, it can be inferred that c for *Linear* goes up to $\sim$12, whereas it is only $\sim$0.05 for *MuVE* and *uMuVE*.

Progressive Results (Fig. 15): In this experiment we demonstrate uMuVE's ability to produce results in a progressive fashion. Particularly, Fig. 15 shows the delay (i.e., processing cost) until producing the i^{th}-top view when recommending a total of 15 views (i.e., top-k, where $k = 15$). As the figure shows, *Linear-Linear*, *MuVE-Linear*, *MuVE-MuVE* and *uMuVE-Linear* produce the top-15 views all together as a batch. However, *uMuVE-uMuVE* produces the 1^{st} top view as soon as it is identified, and then it keeps producing more views in descending order of their utility values. For instance, *uMuVE-uMuVE* recommends the 9^{th} view after only 0.18sec, while *MuVE-MuVE* recommends it along with all the 15 top-k views after 1sec.

Memory Requirements (Fig. 16): In this experiment we study the performance of our memory-aware MuMuVE scheme under a predefined memory constraint M_L. We particularly evaluate the different variants of MuMuVE, namely: Max-bins, Max-utility, and Max-Min-Utility, against uMuVE. To do that, we set the limit M_L as a percentage of the memory used by uMuVE in the worst case (as shown on the x-axis in Fig. 16). For instance, a value of

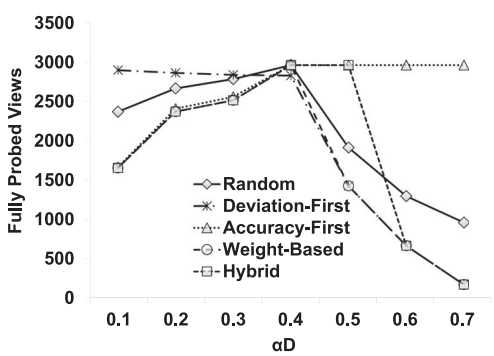


Fig. 13. Priority function analysis.

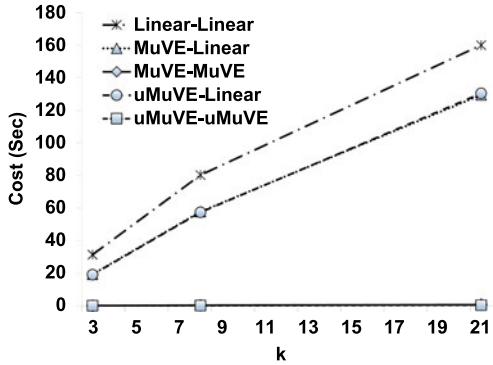


Fig. 14. NBA: Scalability.

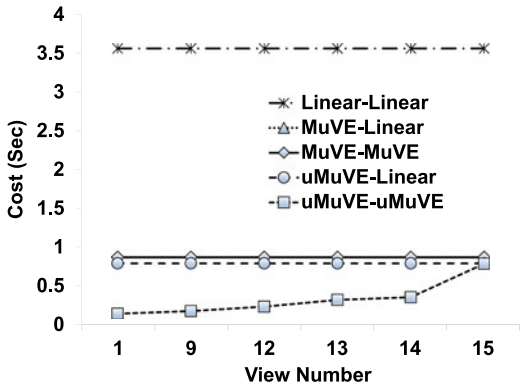


Fig. 15. DIAB: Progressive results.

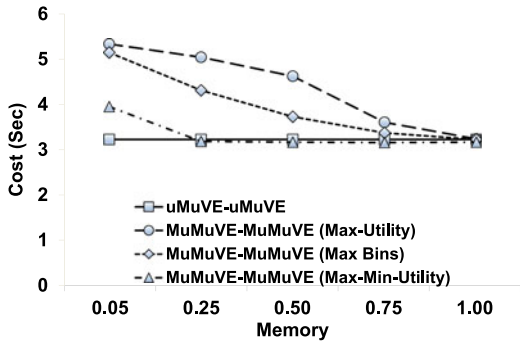


Fig. 16. Impact of M_L on MuMuVE scheme.

0.75 means that the memory constraint M_L is set to 75 percent of the maximum memory used by uMuVE. As the figure shows, *uMuVE-uMuVE* acts as a baseline and its performance is independent of M_L. Further, the figure also shows that as M_L is decreased, the cost of all the memory-aware schemes increase. This is because the memory-aware schemes are forced to evaluate extra objectives to reclaim space. The figure shows that Max-Bins outperforms Max-Utility because of its ability to avoid unnecessary probes and reclaim space, whereas Max-Min-Utility performs as good as *uMuVE-uMuVE* while keeping the memory usage well under constraint.

8 Related Work

Recent years have seen the introduction of many visual analytic tools such as Tableau, Qlik and Spotfire [3], [5], [6]. These tools aim to provide aesthetically high-quality visualizations of the subsets of data selected by an analyst. However, the selection of the interesting attributes and subsets of data remains a user-driven manual task. As such, those tools provide a limited support for automatic data exploration, especially for large data sets.

Alongside the commercial tools discussed above, several research efforts have been directed towards providing automation features to the visualization process. For instance, Rank-by-Feature Framework [27] computes statistical summaries and ranking for histograms and scatter-plots. While visualizations can be ranked by various features, the user still has to select a ranking criterion, and then all possible projections are ranked by that criterion. Profiler [19] detects anomalies and recommends visualizations based on mutual information metric. Hence, it is specifically designed to highlight data quality issues, but exploring data for interesting view is beyond the scope of that work. VizDeck [20] generates all possible 2-D visualizations on a dashboard and allows users to reorder, share or permanently store those visualizations. It also recommend views, based on a visualization quality model using statistical features of the dataset. VizDeck lacks the deviation based ranking and it does not scale for high dimensional large datasets. The ziggy approach [25], [26] introduces a multi-view subset characterization approach based on the idea of selecting tuples that differ from the rest of the database. It recommends sets of columns on which user selected data has an unusual distribution from the rest of the database. Specifically, it performs tuple based ranking, while our work focuses on binned aggregate views.

As mentioned earlier, our work recommends visualizations based on the deviation between two datasets, as in SeeDB [30], [31]. However, while SeeDB effectively recommend views for categorical attributes, it lacks the necessary techniques for handling numerical attributes, which is focus of our work.

9 Conclusions and Future Work

In this paper we presented a novel utility function and a suite of search schemes for recommending top-k aggregate data visualizations. Our utility function recognizes the impact of numerical dimensions on visualization, which is captured by means of multiple objectives, namely: deviation, accuracy, and usability. Our proposed search schemes further incorporate that utility function for the purpose of recommending the top-k aggregate data visualizations. A key goal in the design of those search schemes is to efficiently prune the prohibitively large search space of possible data aggregations. That goal is reasonably achieved by our MuVE scheme, and is further improved by uMuVE, at the expense of a high memory usage. Accordingly, we presented MuMuVE, which provides a pruning power close to that of uMuVE, while keeping memory usage within predefined constraint.

In the future, we plan to extend the data-driven approach adopted in this work to incorporate a user-driven approach for recommending data visualizations. In particular, in our data-driven approach, the interestingness of a view is captured by its deviation from the entire database, irrespective of the user's preferences (with the exception of setting the weights in our multi-objective utility function). Hence, in the future we plan to investigate different methods for capturing different aspects of the user's preference. For instance, we are considering an interactive approach, in which the user is presented with a small set of sample views and they are requested to provide relevance feedback (i.e., if the view is relevant to their analysis task), similar to the approaches proposed in [12], [15]. That feedback is used to build a predictive model to learn the user's preference, which is to be integrated with our data-driven model. Orthogonally, we also plan to extend our problem for the cases where recommendations can be made based on the availability of a history of views that the user has found to be interesting in the past, or that have been identified as interesting by similar users. Towards that we will investigate the integration of collaborative filtering model with our approach.

ACKNOWLEDGMENTS

This work is partially supported by ARC grant LP130100164, NSF CDI award OIA-1028162, and UQ Centennial Scholarship. The authors would also like to thank the anonymous reviewers for their thorough feedback.

REFERENCES

[1] (2014, Jul.). [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes
[2] (2016, Aug.). [Online]. Available: https://www.kaggle.com/uciml/adult-census-income
[3] (2017, Feb.). [Online]. Available: https://www.tableau.com/
[4] (2014, Oct.). [Online]. Available: https://www.basketball-reference.com/
[5] (2017, Feb.). [Online]. Available: https://www.qlik.com/
[6] C. Ahlberg, "Spotfire: An information exploration environment," *ACM SIGMOD Record*, vol. 25, no. 4, pp. 25–29, 1996.
[7] E. Bertini, "Quality metrics in high-dimensional data visualization: An overview and systematization," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 2203–2212, Dec. 2011.
[8] N. Bruno, L. Gravano, and A. Marian, "Evaluating top-k queries over web-accessible databases," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2002, pp. 369–380.
[9] G. Cormode and M. N. Garofalakis, "Histograms and wavelets on probabilistic data," *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 8, pp. 1142–1157, Aug. 2010.
[10] G. Cormode, M. N. Garofalakis, P. J. Haas, and C. Jermaine, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Found. Trends Databases*, vol. 4, no. 1–3, pp. 1–294, 2012.
[11] Q. Cui, M. O. Ward, E. A. Rundensteiner, and J. Yang, "Measuring data abstraction quality in multiresolution visualizations," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 5, pp. 709–716, Sep./Oct. 2006.
[12] K. Dimitriadou, O. Papaemmanouil, and Y. Diao, "AIDE: an active learning-based approach for interactive data exploration," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 2842–2856, Nov. 2016.
[13] H. Ehsan, M. A. Sharaf, and P. K. Chrysanthis, "MuVE: Efficient multi-objective view recommendation for visual data exploration," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2016, pp. 731–742.
[14] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *J. Comput. Syst. Sci.*, vol. 66, no. 4, pp. 614–656, 2003.
[15] X. Ge, Y. Xue, Z. Luo, M. A. Sharaf, and P. K. Chrysanthis, "REQUEST: A scalable framework for interactive construction of exploratory queries," in *Proc. IEEE Int. Conf. Big Data*, 2016, 646–655.
[16] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-*k* query processing techniques in relational database systems," *ACM Comput. Surv.*, vol. 40, no. 4, 2008, Art. no. 11.
[17] Y. E. Ioannidis, "The history of histograms (abridged)," in *Proc. 30th Int. Conf. Very Large Data Bases*, 2003, pp. 19–30.
[18] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel, "Optimal histograms with quality guarantees," in *Proc. 30th Int. Conf. Very Large Data Bases*, 1998, pp. 275–286.
[19] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer, "Profiler: Integrated statistical analysis and visualization for data quality assessment," in *Proc. Int. Working Conf. Adv. Vis. Interfaces*, 2012, pp. 547–554.
[20] A. Key, B. Howe, D. Perry, and C. R. Aragon, "VizDeck: Self-organizing dashboards for visual analytics," in *Proc. ACM SIGMOD Int. Conf. Manag. Data*, 2012, pp. 681–684.
[21] H. A. Khan and M. A. Sharaf, "Progressive diversification for column-based data exploration platforms," in *Proc. IEEE 25th Int. Conf. Data Eng.*, 2015, pp. 327–338.
[22] H. A. Khan, M. A. Sharaf, and A. Albarrak, "DivIDE: Efficient diversification for interactive data exploration," in *Proc. Int. Conf. Scientific Statistical Database Manage.*, 2014, Art. no. 15.
[23] Z. Liu, B. Jiang, and J. Heer, "imMens: Real-time visual querying of big data," *Comput. Graph. Forum*, vol. 32, no. 3, pp. 421–430, 2013.
[24] A. Marian, N. Bruno, and L. Gravano, "Evaluating top-*k* queries over web-accessible databases," *ACM Trans. Database Syst.*, vol. 29, no. 2, pp. 319–362, 2004.
[25] T. Sellam and M. L. Kersten, "Fast, explainable view detection to characterize exploration queries," in *Proc. Int. Conf. Scientific Statistical Database Manage.*, 2016, Art. no. 20.
[26] T. Sellam and M. L. Kersten, "Ziggy: Characterizing query results for data explorers," *Proc. VLDB Endowment*, vol. 9, no. 13, pp. 1473–1476, 2016.
[27] J. Seo and B. Shneiderman, "Knowledge discovery in high-dimensional data: Case studies and a user survey for the rank-by-feature framework," *IEEE Trans. Vis. Comput. Graph.*, vol. 12, no. 3, pp. 311–322, May./Jun. 2006.
[28] C. Stolte, D. Tang, and P. Hanrahan, "Polaris: A system for query, analysis, and visualization of multidimensional relational databases," *IEEE Trans. Vis. Comput. Graph.*, vol. 8, no. 1, pp. 52–65, Jan./Mar. 2002.
[29] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, "Online outlier detection in sensor data using non-parametric models," in *Proc. 30th Int. Conf. Very Large Data Bases*, 2006, pp. 187–198.
[30] M. Vartak, S. Madden, A. G. Parameswaran, and N. Polyzotis, "SEEDB: Automatically generating query visualizations," in *Proc. VLDB Endowment*, vol. 7, no. 13, pp. 1581–1584, 2014.
[31] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis, "SEEDB: efficient data-driven visualization recommendations to support visual analytics," *Proc. VLDB Endowment*, vol. 8, no. 13, pp. 2182–2193, 2015.

Humaira Ehsan received the BS degree in computer science from NUFAST, and the MS degree in computer science from Lahore University (LUMS), Pakistan. She is currently working towards the PhD degree in computer science with the University of Queensland. Her research interests include database systems with a focus on data exploration and visualization.

Mohamed A. Sharaf received the PhD degree in computer science from the University of Pittsburgh, in 2007, and was a postdoctoral research fellow with the University of Toronto until 2009. He is a senior lecturer in the School of ITEE and a member of the DKE group with The University of Queensland. His research interests include the general area of database management systems, with special emphasize on data exploration and visual analytics.

Panos K. Chrysanthis received the BS degree from the University of Athens, Greece, the MS and the PhD degrees from the University of Massachusetts at Amherst, in 1982, 1986, and 1991, respectively. He is a professor of computer science and the founding director of the Advanced Data Management Technologies Laboratory, the University of Pittsburgh. He is also an adjunct professor with Carnegie Mellon University. His research interests include lie at the intersection of data management, distributed systems, and collaborative applications. He is a recipient of the US NSF CAREER Award, an ACM Distinguished Scientist, and a senior member of the IEEE. In 2018, he will be the TPC co-chair of the IEEE ICDE.