MuVE: Efficient Multi-Objective View Recommendation for Visual Data Exploration

Humaira Ehsan, Mohamed A. Sharaf
School of Information Technology and Electrical Engineering
The University of Queensland, Queensland, Australia
{h.ehsan, m.sharaf}@uq.edu.au

Panos K. Chrysanthis
Department of Computer Science
University of Pittsburgh, Pennsylvania, USA
panos@cs.pitt.edu

***Abstract*—To support effective data exploration, there is a well-recognized need for solutions that can automatically recommend interesting visualizations, which reveal useful insights into the analyzed data. However, such visualizations come at the expense of high data processing costs, where a large number of views are generated to evaluate their usefulness. Those costs are further escalated in the presence of numerical dimensional attributes, due to the potentially large number of possible binning aggregations, which lead to a drastic increase in the number of possible visualizations. To address that challenge, in this paper we propose the *MuVE* scheme for Multi-Objective View Recommendation for Visual Data Exploration. MuVE introduces a hybrid multi-objective utility function, which captures the impact of binning on the utility of visualizations. Consequently, novel algorithms are proposed for the efficient recommendation of data visualizations that are based on numerical dimensions. The main idea underlying MuVE is to incrementally and progressively assess the different benefits provided by a visualization, which allows an early pruning of a large number of unnecessary operations. Our extensive experimental results show the significant gains provided by our proposed scheme.**

I. Introduction

Data visualization is gaining growing interest as an indispensable tool for data exploration and analysis in a widely diverse set of discovery-oriented applications [9], [23]. In such applications, data analysts explore large volumes of data looking for interesting visualization that reveal new and valuable insights. Such process is typically ad-hoc and labor-intensive, especially for high-dimensional databases. Motivated by the need for efficient data analysis and exploration, several solutions for recommending visualizations have recently emerged to guide analysts throughout that rather time-consuming process (e.g., [22], [21], [14], [13]).

The main idea underlying those solutions is to automatically generate all possible views of data, and *recommend* the *top-k interesting* views, where an interestingness of a view 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 [22]. In particular, the deviation-based metric measures the distance between the probability distribution of the specific dataset under analysis and that of a comparison dataset, which is typically the entire database from which that dataset is extracted. The underlying premise is that a visualizations that results in a higher deviation is expected to reveal some interesting insights that are very particular to the analyzed dataset [22], [21].

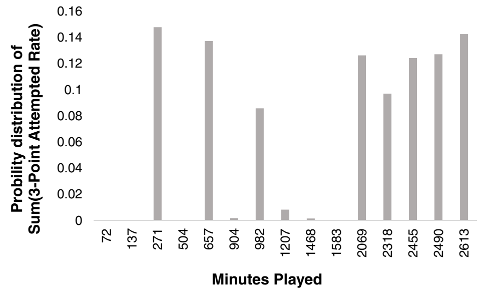


Fig. 1: View on players of the GSW team (target view)

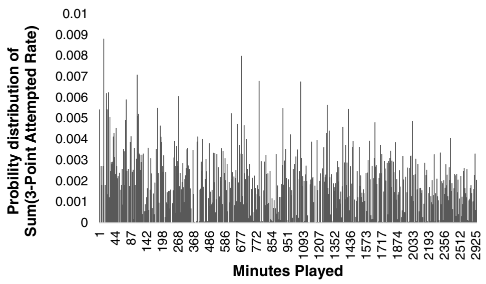


Fig. 2: View on all players in the 2015 NBA (comparison view)

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, *binnied aggregation* is typically required so that to group the numerical values along a dimension into adjacent intervals [6], [11]. 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

Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 21:29:49 UTC from IEEE Xplore. Restrictions apply.

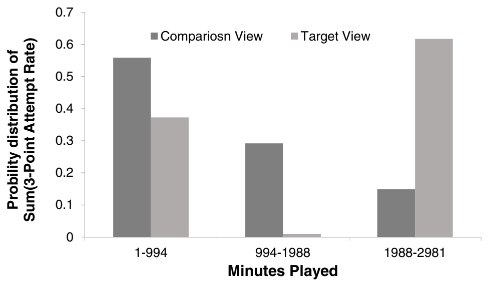


Fig. 3: Binned target view (i.e., GSW team) and comparison view (i.e., all NBA teams)

(GSW) basketball team to win the 2015 NBA championship. Consequently, the analyst uses the 2015 NBA players statistics database [2] 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. Figure 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 Figure 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 Figure 3 shows some very interesting observation. Particularly, Figure 3 shows that for all NBA players, the `3PAr` decreases as they play more games throughout the season. Intuitively, this is perfectly understandable since the fatigue incurred from playing more games can affect their fitness and reduce their `3PAr`. However, for the GSW players, that pattern is very different from that general pattern. As Figure 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 players in other teams. Clearly, that observation reflects the high fitness and consistency of the GSW players, which distinguishes them from the 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 if 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], [3]. 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., [6], [11], [12]). 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 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, in this work, we propose the *MuVE* scheme for **Mu**lti-Objective **V**iew Recommendation for Visual Data **E**xploration. In MuVE, 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 those often conflicting objectives dramatically expands the search space of possible visualizations. Moreover, it significantly increases the processing time incurred to asses the overall utility of each visualization, which is assembled from the utility values of each of the three objectives listed above.

Accordingly, in MuVE, we propose a suite of novel search algorithms, which are particularly optimized to leverage the specific features of the view recommendation problem. The main idea underlying MuVE is to use an optimized incremental evaluation of the multi-objective utility function, in which the different objectives are computed progressively as needed. This techniques allows for pruning a large number of unnecessary views (i.e., low utility views), and in turn reduce the overall processing time incurred for recommending the top-k views. The main contributions of this work are as follows:

- We formulate and analyze the problem of recommending visualizations in the presence of numerical attribute dimensions (**Section III**).
- We introduce a novel hybrid multi-objective utility function, which captures the impact of binning numerical dimensions on the overall utility of recommended visualizations (**Section III**).
- We propose the MuVE scheme, which introduces a novel search algorithm that is particularly optimized to leverage the specific features of the view recommendation problem (**Section IV**).
- We extend the MuVE scheme with further approximations, which allow for significant improvement in performance while maximizing the fidelity of the recommendations (**Section IV**).
- We conduct extensive experimental evaluation on real datasets, which illustrate the benefits achieved by MuVE (**Sections V and VI**).

Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 21:29:49 UTC from IEEE Xplore. Restrictions apply.

II. Background and Related Work

A. 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 [22], [21], [19], [14], [13], our model is based on a multi-dimensional database D_B, consisting of a set of dimension attributes $\mathbb{A}$ and a set of measure attributes $\mathbb{M}$. Additionally, $\mathbb{F}$ is the set of possible aggregate functions over the measure attributes $\mathbb{M}$, such as SUM, COUNT, AVG, STD, VAR, MIN and MAX. Hence, an aggregate view V_i over D_Q is represented by a tuple (A, M, F) where $A \in \mathbb{A}$, $M \in \mathbb{M}$, and $F \in \mathbb{F}$. That is, D_Q is grouped by dimension attribute A and aggregated by function F on measure attribute M. As in [22], we considers aggregate views that perform a single-attribute group-by and aggregation on D_Q. Accordingly, 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, manually finding those interesting and useful views of data is a time-consuming, cumbersome task. The complexity of this task stems from: 1) the large number of possible visualizations, and 2) the interestingness of a visualization is rather subjective. In particular, an analyst typically explores a database with some vague idea of what she is looking for, where insights become only clear in hindsight. Towards automated visual data exploration, recent approaches have been proposed for recommending interesting visualizations based on some objective, well-defined quantitative metrics (e.g., [22], [21], [14], [13]). 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 [22].

In particular, the deviation-based metric measures the distance between $V_i(D_Q)$ and $V_i(D_B)$. That is, it measures the deviation between the aggregate view V_i generated from the subset data D_Q vs. that generated from the entire database D_B, where $V_i(D_Q)$ is denoted as *target* view, whereas $V_i(D_B)$ is denoted as *comparison* view. The premise underlying the deviation-based metric is that a view V_i that results in a higher deviation is expected to reveal some interesting insights that are very particular to the subset D_Q and distinguish it from the general patterns in D_B.

To ensure that all views have the same scale, each target view $V_i(D_Q)$ is normalized into a *probability distribution* $P[V_i(D_Q)]$ and each comparison view into $P[V_i(D_B)]$. For example consider an aggregate view $V = (A, M, F)$. The result of that view can be represented as the set of tuples: $< (a_1, g_1), (a_2, g_2), ..., (a_t, g_t) >$, where t is the number of distinct values (i.e., groups) in attribute A, a_i is the i-th 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 normalized probability distribution $P[V] = < \frac{g_1}{G}, \frac{g_2}{G}, ..., \frac{g_t}{G} >$.

For instance, in Example 1, Figure 1 shows a probability distribution target view over the GSW team players (i.e., $P[V_i(D_Q)]$), whereas Figure 2 shows the corresponding probability distribution comparison view over all the players in the 2015 NBA database (i.e., $P[V_i(D_B)]$).

Accordingly, the deviation $D(V_i)$, provided by a view V_i, is defined as the distance between those two probability distributions. Formally, for a given distance function $dist$ (e.g., Euclidean distance, Earth Mover's distance, K-L divergence, etc.), $D(V_i)$ is defined as:

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

Consequently, the deviation $D(V_i)$ of each possible view V_i is computed, and the k views with the highest deviation are recommended (i.e., *top-k*) [22]. 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, such as shared computation among views, early pruning of low-deviation views, and sampling (e.g., [23], [22], [15]). Such optimization techniques are general enough to be incorporated in the backend database engine of any data visualization platform, and are orthogonal to the optimizations proposed in this work to address the impact of numerical dimensions, which is described next.

B. 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 [6], [11]. 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

Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 21:29:49 UTC from IEEE Xplore. Restrictions apply.

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 Figures 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., [6], [11], [12]), anomaly detection (e.g., [20]), and data visualization (e.g., [13], [16]). 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. Such histograms provide a concise summary of the underlying data distribution of an attribute (i.e., frequency of attribute values). While a histogram that is based on a small number of bins provides a high degree of compression, its accuracy can be quite poor. To the contrary, adding more bins to a histogram, or equivalently decreasing its bin width, has been shown to increase the accuracy of a histogram at the expense of increasing the costs incurred in processing, maintaining, and storing those additional bins. Similarly, in bar chart visualizations, which is the focus of this paper, too few bins result in loss of information and compromise the accuracy of visualization, while too many bins result in a cluttered low quality visualization.

Deciding the optimal bin width (or number of bins) for histograms has been intensively studied in the statistics literature, where several *model-based* approaches have been proposed [6]. 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 [6], [11], for comprehensive surveys on that topic.) In this work, we adopt the same approach and expand on existing model-free database methods.

III. Multi-objective View Recommendation

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 [22], [21], as described in Section II-A. However, that simple notion of utility falls short in capturing the impact of numerical dimensions. In particular, the presence of numerical dimensions further introduces additional factors that impact the utility gained from a recommended view, as described next.

A. 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, Figure 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, and query optimization 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, for a binned view such as view $V_{i,b}$ defined above, its usability in terms of visual quality and clutter reduction [3], [7], is simply quantified in terms of the relative bin width, which is defined as follows:

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

where $S(V_{i,b})$ 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: $<(a_1, g_1), (a_2, g_2), ..., (a_j, g_i), ..., (a_t, g_t)>$, 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, $V_{i,b} =< (I_1, \hat{g}_1), (I_2, \hat{g}_2), ..., (I_x, \hat{g}_x), ..., (I_b, \hat{g}_b) >$, 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. A widely used metric for measuring that kind of (in)accuracy is the Sum Squared Error (SSE), which

Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 21:29:49 UTC from IEEE Xplore. Restrictions apply.

has also been employed in the context of frequency histograms [12], [5]. Applying the same 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 follows:

$$E(V_{i,b}) = \sum_{p=1}^{t} (g_p - g'_p)^2$$

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:

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

where $A(V_{i,b})$ 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.

B. Problem Definition

In our proposed scheme MuVE, 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 under analysis, 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 (i.e., distribution) 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. Alternatively, these can be system-defined and are set automatically to meet certain objectives that are defined by the application. Setting any of those weights to zero implies that the user is indifferent to the objective associated with that weight. For instance, setting both $\alpha_A = 0$ and $\alpha_S = 0$ is equivalent to basic definition of utility, which is only based on deviation. Meanwhile, setting $\alpha_D = \alpha_A = \alpha_S$ implies the user prefers to give equal weight to all three objectives. 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 I, while Figure 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, Figure 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 MuVE recommendation list (i.e., *top-1*), and enabled for an insightful visualization of the analyzed data.

C. View Processing Cost

Recall that in the absence of numerical dimensions (i.e., only categorical dimensions are considered), 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.

In the presence of numerical dimensions, the search space (i.e., number of candidate views) is further expanded due to the different binning options. Moreover, the complexity of the operations performed on those views is also increased due to incorporating additional objectives. In particular, for each

Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 21:29:49 UTC from IEEE Xplore. Restrictions apply.

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:

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

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: Is the time required to process the raw data to generate the candidate target and comparison binned views, where the cost for generating the target view is denoted as $C_t(V_{i,b})$, and that for generating the comparison view is denoted as $C_c(V_{i,b})$.
- Deviation Computation Time: Is the time required to measure the deviation between the target and comparison binned views, and is denoted as: $C_d(V_{i,b})$. Notice that this time depends on the employed distance function $dist$.
- Accuracy Evaluation Time: Is 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:

$$C(V_i) = \sum_{b=1}^{B} C_t(V_{i,b}) + C_c(V_{i,b}) + C_d(V_{i,b}) + C_a(V_{i,b}) \quad (6)$$

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

$$C = \sum_{i=1}^{N_B} C(V_i) \quad (7)$$

In the next section, we describe our efficient MuVE scheme for minimizing that cost C, while maintaining the quality of recommendation.

IV. Search Strategies

In this section, we present search strategies for finding the top-k binned views selected 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 Figure 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 IV-A, we present different strategies for horizontal search, including our optimized MuVE scheme, whereas in Section IV-B, we expand on those strategies to enable and integrate vertical search. Finally, in Section IV-C, we present approximation techniques that further increase the cost savings provided by MuVE, while maintaining the fidelity of recommendation comparable to the optimal.

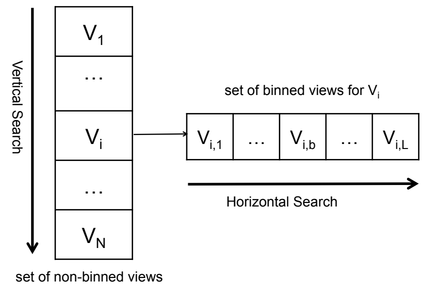


Fig. 4: Horizontal and Vertical Searches for recommending top-k visualizations

A. Horizontal Search

As discussed in the previous section, for a non-binned non-binned view V_i over a numerical dimension A_j, the number of possible target and comparison binned views is equal to: $2 \times |\mathbb{M}| \times |\mathbb{F}| \times B_j$. 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 our proposed MuVE scheme for finding such optimal binning, together with two baseline schemes, namely 1) Linear Search (optimal baseline), and 2) Local Search (approximate baseline). We defer our discussion of vertical search algorithms to Section IV-B.

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: $\mathbb{V}_i = \{V_{i,1}, V_{i,2}, ..., V_{i,b}, ..., V_{i,L}\}$, 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 $V_{i,opt}$.

2) Local Search: Local search is a metaheuristic method that is widely used in solving optimization problems. In general, a local search algorithm starts out with an initial solution and then attempts to find a better solution in the neighborhood of that initial one. In this work, we adopt dynamic *Hill Climbing (HC)*, with halving search as another baseline method [4]. Specifically, for the set of binned views $\mathbb{V}_i = \{V_{i,1}, V_{i,2}, ..., V_{i,L}\}$, HC initially starts at some random number of bins b, where $1 \leq b \leq L$, and a step $s = L$. In each iteration of HC, it considers two alternative settings for the number of bins: $b - s$, and $b + s$, then moves to the one which provides maximum utility. When HC cannot find a

Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 21:29:49 UTC from IEEE Xplore. Restrictions apply.

move that increases the utility, then s is halved. This halving continues until $s < 1$. Despite of being susceptible to hitting local maxima, HC is expected to provide significant reductions in processing costs compared to the linear search methods.

3) 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: $\mathbb{V}_i = \{V_{i,1}, V_{i,2}, ..., V_{i,b}, ..., V_{i,L}\}$. 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., [17], [8]), are not directly applicable to our problem because firstly 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, secondly 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)* [8]. 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 $\alpha_S S(V_{i,1}), \alpha_S S(V_{i,2}), ..., \alpha_S S(V_{i,L})$, 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 [17], [8], [10]. 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 [17], [8]): 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 by means of 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. To decide the evaluation oder of those two objectives, MuVE applies a simple priority function, such that if:

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

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

Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 21:29:49 UTC from IEEE Xplore. Restrictions apply.

those costs over the previous $V_{i,1}, V_{i,2}, ..., V_{i,b-1}$ 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.

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 binned views starting at $V_{i,b+1}$ will have lower S values, then they are also guaranteed to provide utilities less than U_{seen}. Hence, those views are pruned and an early termination is reached.

B. 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 $N_B = \sum_{j=1}^{|\mathbb{A}|} 2 \times |\mathbb{M}| \times |\mathbb{F}| \times B_j$. Hence, 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 to perform both the vertical and horizontal searches (i.e., MuVE-MuVE).

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 $\alpha_S S(V_{i,1}), \alpha_S S(V_{i,2}), ..., \alpha_S S(V_{i,L})$, 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.

C. MuVE Approximations

All the search algorithms presented so far, except for Hill Climbing, are accurate in the sense that they provide the same top-k views as the baseline exhaustive linear search. Meanwhile, Hill Climbing, being a local search algorithm, is prone to hit some local maxima when used for horizontal search, hence recommending views with lower utility. The degree of inaccuracy exhibited by local search methods is typically unpredictable and highly depends on the behavior of the utility function to be optimized. To the contrary, MuVE employs optimization techniques that allow for significant performance gains, while at the same time providing the same recommendations as the exhaustive baseline. In the following, we introduce several approximations to the MuVE scheme to further improve its performance, while incurring negligible loss in the quality of recommendation.

1) View Refinement: In this approximation, instead of horizontally expanding and searching each and every non-binned view V_i to find its optimal binned view $V_{i,opt}$, only a small number of views are selected for that expensive horizontal search. The main idea is to perform a simple vertical search to quickly select that small set of views, which are then refined using horizontal search to find the top-k recommended views. To achieve this, the set of non-binned views $\mathbb{V}$ is searched vertically using a predefined number of bins, which is the same

Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 21:29:49 UTC from IEEE Xplore. Restrictions apply.

for all views. Specifically, the utility of each non-binned view $V_i \in \mathbb{V}$ is computed as $U(V_{i,def})$, where def is the same value for all views. Then the top-k views according to that binning def are selected, which is easily performed using linear search or MuVE. Consequently, utility of each one of those k selected views (i.e., $U(V_{i,def})$) is further refined using horizontal search to find its optimal binning (i.e., $V_{i,opt}$). Hence, only k views are selected for horizontal search, which can be applied using linear search, MuVE, or HC, as explained earlier. We note that the default binning value def is a system parameter. Our experimental evaluation shows that choosing a moderate number of bins results in significant reductions in cost, while at the same time providing high fidelity recommendations.

2) View Skipping: The main idea underlying that approximation is to skip the horizontal search for some non-binned views, thus saving the costs incurred in finding their $V_{i,opt}$. To better understand this idea, recall that each non-binned view V_i is basically represented by a tuple (A, M, F) where $A \in \mathbb{A}$, $M \in \mathbb{M}$, and $F \in \mathbb{F}$. Hence, for each numerical dimension A, there exists a set of views $\mathbb{V}_A$, which share the same dimension A, while being defined using different measures and aggregate functions, such that $|\mathbb{V}_A| = |\mathbb{M}| \times |\mathbb{F}|$. Accordingly, in this approximation we assign the same binning to all the views in the set $\mathbb{V}_A$. To find that binning (i.e., number of bins), a view V_i is selected arbitrary from the set $\mathbb{V}_A$. For that non-binned view, horizontal search is performed normally using any of the strategies outlined above (i.e., linear, HC, or MuVE), so that to find its corresponding $V_{i,opt}$. Then that optimal number of bins *opt* is assigned to all views in $\mathbb{V}_A$, and their utilities are evaluated accordingly. The premise is that the range of a numerical dimension A is an important factor in deciding its optimal binning. Hence, since all views in $\mathbb{V}_A$ share the same dimension A, then the optimal binning of those views will have a very small variance from some mean value, which is selected as described above and used to represent the set of views $\mathbb{V}_A$. Accordingly, the number of times horizontal search is invoked is equal to the number of numerical dimensions (i.e., $|\mathbb{A}|$). Those horizontal searches are easily integrated with one of the vertical searches described above (i.e., linear or MuVE).

3) Range Partitioning: The idea for this approximation is to reduce the complexity of the horizontal search based on simple range partitioning techniques. Recall that a non-binned view $V_i = (A, M, F)$ is expanded into a set of binned views: $\mathbb{V}_i = \{V_{i,1}, V_{i,2}, ..., V_{i,b}, ..., V_{i,L}\}$, where b is the number of bins, and L is the range of the continuous numerical dimension A. By default, MuVE as well as linear search, assume the value of b to be continuous in the range $[1-L]$ with incremental additive step of 1.0, leading to corresponding binning of widths: $\frac{L}{1}, \frac{L}{2}, ..., 1$. Hence, a numerical dimension A with a large continuous range results in a large number of binning options, and in turn a high cost for performing horizontal search. Accordingly, we employ two simple alternative methods for range partitioning, namely: 1) additive, and 2) geometric. The default partitioning described above is an instant of the additive method, in which the incremental step s is set to 1. In the general case, the incremental step s is a parameter, hence, a non-binned view V_i is expanded into a set of binned views: $\mathbb{V}_i = \{V_{i,1}, V_{i,1+s}, V_{i,1+2s}, ..., V_{i,L}\}$. Alternatively, in the geometric method (e.g., [18]), V_i is expanded into a set of binned views: $\mathbb{V}_i = \{V_{i,2^0}, V_{i,2^1}, V_{i,2^2}, ..., V_{i,L}\}$. Naturally, both methods are expected to reduce the processing time incurred in horizontal search, at the expense of some reduction in the fidelity of recommendation, which will be evaluated experimentally in the next sections.

V. Experimental Testbed

We perform extensive experimental evaluation to measure both the efficiency and effectiveness 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 [22] 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. This leads to the following combinations: *Linear-Linear*, *HC-Linear*, *MuVE-Linear*, and *MuVE-MuVE*. For instance, in *MuVE-Linear*, MuVE is used for horizontal search, whereas linear search is applied for vertical search. In the presence of approximation, as discussed in Section IV, we extend our notation to: *SearchH(AppH)-SearchV(AppV)*. Hence, the possible horizontal approximations are: *SearchH(A)*, and *SearchH(G)*, which denote the additive and geometric range partitioning, respectively. For vertical approximations, *SearchV(R)* denotes the view refinement approximation, and *SearchV(S)* is for the view skipping approximation.

Data Analsyis: We assume a data exploration setting in which multi-dimensional datasets are analyzed. We use two datasets: *DIAB:* dataset of diabetic patients [1] and *NBA:* dataset of basketball players [2]. The DIAB dataset has 9 attributes and 768 tuples. The independent numeric attributes of the dataset are used as dimensions (e.g., age, BMI, etc.), whereas the observation attributes are used as measures (insulin level, glucose concentration, etc.). In our default setting, we select 3 dimensions, 3 measures, and 3 aggregate functions, which results in a maximum of 2961 possible views. The NBA dataset has 28 attributes and 651 tuples. Similar to DIAB, dimensions are selected from independent numeric attributes (e.g., age, number of minutes played, etc.), while measures are selected from observations (e.g., player efficiency rating, 3-point attempt rate, etc.). Also, as in the DIAB dataset, we experiment with 3 dimensions, and 3 aggregate functions, while the number of measures vary between 3 and 13, with 3 being the default, which results in a maximum of 27,756 possible views. In the analysis, all the α values are in the range $[0-1]$, where $\alpha_D + \alpha_A + \alpha_S = 1$. In the default setting, $\alpha_D = 0.2$, $\alpha_A = 0.2$, $\alpha_S = 0.6$, $k = 5$, and euclidean distance is used for measuring deviation, unless specified otherwise.

Performance: We evaluate the efficiency and effectiveness of the different recommendations strategies in terms of the following two metrics: (1) *Cost:* As mentioned in Section III,

Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 21:29:49 UTC from IEEE Xplore. Restrictions apply.

the cost of a strategy is the total cost incurred in processing all the candidate binned views. We use wall clock time to measure the different components included in that cost namely, query execution time of target and comparison views, deviation computation time, and accuracy evaluation time. Each setting is executed 10 times and then average is taken as the cost incurred; and (2) *Fidelity:* It is a measure of the degree of accuracy achieved by a certain scheme. Particularly, if $\mathbb{V}_{opt}$ is the set of top-k views recommended by a baseline optimal scheme, whereas $\mathbb{V}_{rec}$ is the set of views recommended by an approximated scheme, then fidelity is measured to capture the difference between the sum of the utilities offered by $\mathbb{V}_{opt}$ and $\mathbb{V}_{rec}$. Formally: $F = 1 - \frac{U(\mathbb{V}_{opt}) - U(\mathbb{V}_{rec})}{U(\mathbb{V}_{opt})}$

VI. Experimental Evaluation

In the following experiments, we evaluate the performance of both our optimization techniques (Section VI-A), as well as our approximation techniques (Section VI-B), under different parameter settings.

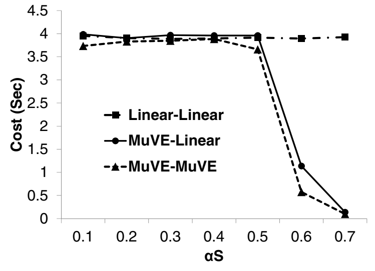
A. Optimization Techniques

Impact of the α parameters (Figures 5 and 6): In this set of experiments, we measure the impact of the α values on processing time (i.e., cost). Figures 5 and 6 show how the cost of the *Linear-Linear*, *Linear-MuVE* and *MuVE-MuVE* schemes is affected by changing the values of α_D, α_A and α_S. Notice that no approximations are employed in those schemes, hence, the fidelity of all schemes is 100%.

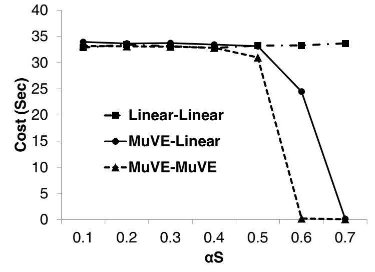
In Figure 5, α_A is set to constant 0.2 while α_S and α_D are changing. In particular, as shown in the figure, α_S is increased, while α_D is implicitly decreased and is easily computed as $\alpha_D = 1 - \alpha_S + \alpha_A$. Figures 5a and 5b show that *Linear-Linear* has almost same cost for all values of α_S for both datasets, which is expected since it performs exhaustive search over all combinations of dimensions, measures, aggregate functions, and number of bins. Therefore, its cost depends on the number of all possible combinations, irrespective of the values of α.

Figures 5a and 5b also show that *MuVE-Linear* and *MuVE-MuVE* 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 Figure 5a at $\alpha_S > 0.5$, both schemes show more than 70% reduction in cost as compared to the *Linear-Linear*. This happens because in the *MuVE* 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 pruning is further increased under *MuVE-MuVE*, which is able to prune those operations during both the vertical and horizontal searches. For instance, Figure 5b shows that *MuVE-MuVE* reduces the processing cost by almost 70%, compared to *MuVE-Linear*, at $\alpha_S = 0.6$. *MuVE-MuVE* shows more reduction in cost for the NBA dataset as compared to the DIAB dataset because the NBA dataset has dimension attributes with larger ranges, which results in large number of pruned views.

In Figure 6, $\alpha_S = 0.2$, whereas α_D is increasing and accordingly α_A is decreasing. Figures 6a and 6b show the effect of changing α values on cost. Particularly, Figure 6a



(a) DIAB: Cost



(b) NBA: Cost

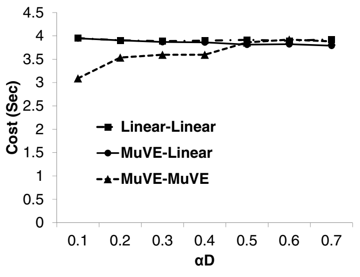
Fig. 5: Impact of α_D and α_S on cost, while $\alpha_A = 0.2$

shows that *MuVE-MuVE* offers the lowest cost, 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. This can be further understood using Figure 6c, in which we plot the number of views that are probed in full (i.e., both deviation and accuracy are evaluated). Figure 6c also shows that *MuVE-MuVE* fully probes 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 (Figure 6a). This is because at high α_D, *MuVE-MuVE* mainly prunes the operations for computing accuracy, whereas at high α_A it mainly prunes the operations for computing deviation, which incurs higher processing cost than that needed for computing accuracy.

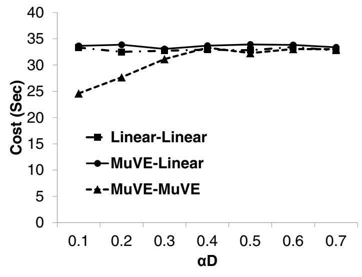
Impact of k (Figure 7): In the previous experiments, the value of k is set to 5 (i.e., top-5 views are recommended). Figure 7 shows that *Linear-Linear* and *MuVE-Linear* are both insensitive to the increase in the value of k. This is because *Linear-Linear* is exhaustive search, whereas *MuVE-Linear* also performs an exhaustive vertical search. In *MuVE-MuVE* scheme, as soon as it has seen the top-k highest utility views, early termination will be enabled leading to pruning many unnecessary low utility views, and saving their processing time. For instance, in case of top-1 *MuVE-MuVE* reduces cost by up to 90% compared to the *Linear-Linear* scheme.

Scalability (Figure 8): From Section III-C, 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* and *MuVE*, in practice, however, c is much smaller for *MuVE* due to pruning. For example, Figure 8 shows our results on the

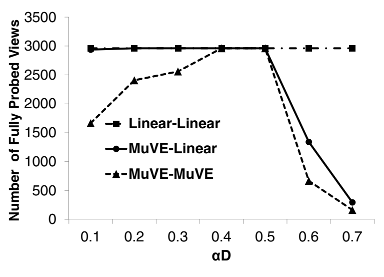
Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 21:29:49 UTC from IEEE Xplore. Restrictions apply.



(a) DIAB: Cost



(b) NBA: Cost



(c) DIAB: Fully Probed Views

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

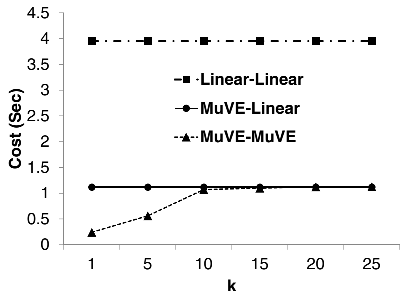


Fig. 7: DIAB: Impact of k on Cost

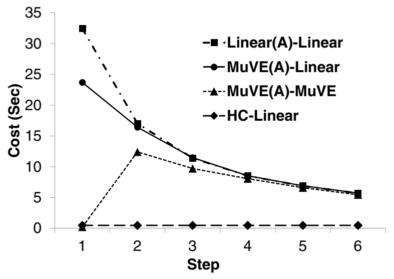


Fig. 9: NBA: Impact of additive range partitioning on cost

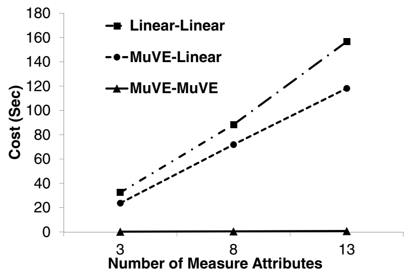


Fig. 8: NBA: Scalability

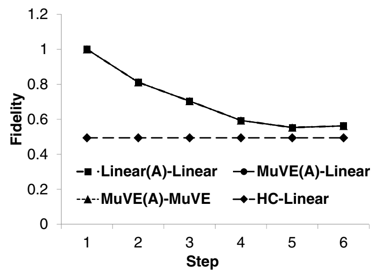


Fig. 10: NBA: Impact of additive range partitioning on fidelity

NBA data, it can be inferred from this graph that *c* for *Linear* goes up to ∼12, whereas it is only ∼0.05 for *MuVE*.

B. Approximation Techniques

Impact of Additive Range Partitioning (Figures 9 and 10): In Figures 9 and 10 we show the impact of having different values of $step$. As expected, Figure 9 shows that the *HC-Linear* search scheme provides the same cost regardless of the employed step. This is simply because *HC* employes its own stepping method, as explained earlier. Meanwhile, the cost of *Linear(A)-Linear* decreases with the increase in $step$. This is because when $step > 1$, the search space is reduced by a factor of $step$, which results in that reduction of cost. The figure also shows that *MuVE(A)-Linear* has low cost at $step = 1$ and after that its cost is almost the same as *Linear(A)-Linear*. This is because, when $step = 1$ *MuVE(A)-Linear* gets the opportunity of short circuits and early terminations, which are activated because of the high utility provided by those views with relatively small number of bins. With higher values of $step$, this opportunity is missed and *MuVE(A)-Linear* behaves almost the same as *Linear(A)-Linear*. Figure 10 shows the impact of $step$ on fidelity. For *HC-Linear*, similar to its cost, its fidelity is insensitive to $step$ and it always provides less than 50% fidelity. This is because *HC-Linear* is a local search based scheme, which is expected to hit a local maxima, hence, failing to achieve the global maxima, which results in low fidelity. Meanwhile, *Linear(A)-Linear*, *MuVE(A)-Linear* and *MuVE(A)-MuVE* show the same pattern with increasing the $step$ value.

Impact of Geometric Partitioning (Figure 11 and 12): For this set of experiments we set $\alpha_A = 0.2$ and observe the effect of changing α_S on the fidelity and cost of when employing geometric partitioning. Figure 11 shows that the cost of *Linear(G)-Linear* remains constant because it exhaustively searches for maximum utility view. Meanwhile, geometric partitioning reduces the cost of *MuVE(G)-Linear* and *MuVE(G)-MuVE* by more than 50% compared to *Linear(G)-Linear* for high values of α_S. In Figure 12, we can see that the fidelity of *HC-Linear* is decreasing with increase in α_S while all other schemes have around 100% fidelity. This is because the geometric

Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 21:29:49 UTC from IEEE Xplore. Restrictions apply.

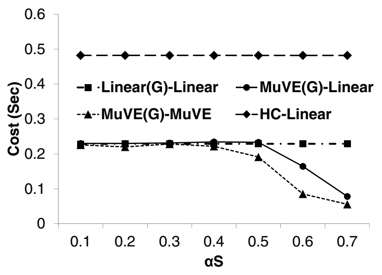


Fig. 11: NBA: Impact of geometric range partitioning on cost

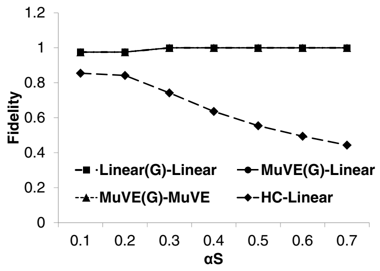


Fig. 12: NBA: Impact of geometric range partitioning on fidelity

partitioning always includes views with small/medium number of bins (e.g., $2^0, 2^1, 2^2, 2^3$), which are typically the ones that provide high overall utility.

Approximation with View Refinement and Skipping (Figure 13): Figure 13 shows the cost of *Linear-Linear(R)* and *Linear-Linear(S)*, which employ approximations during the vertical search. As the figure shows, the cost of *Linear-Linear(S)* is lower than *Linear-Linear* because it reduces the search space by assuming that one bin size for a dimension would fit for all measures and functions. Meanwhile, *Linear-Linear(R)*, with default binning $def = 4$, offers the lowest cost as it vertically choses the top-k views in an initial pass and then the horizontal search is done only for those top-k views. In terms of fidelity, both *Linear-Linear(S)* and *Linear-Linear(R)* achieve around 95% fidelity.

VII. Conclusions

Motivated by the need for supporting visualization recommendation in the presence of numerical dimensions, in this paper we proposed the MuVE scheme for view recommendation. MuVE recognizes the impact of numerical dimensions and employ a multi-objective utility function, which captures that impact in terms of deviation, accuracy, and usability. Consequently, we propose efficient schemes for recommending the top-k visualization that maximize that utility. Additionally, we present approximation techniques that further increase the cost savings provided by MuVE, while maintaining its fidelity. Our experimental results show that employing the MuVE scheme for both binning setting (i.e., horizontal search) and top-k recommendation (i.e., vertical search) offers significant reduction in terms of data processing costs.

Acknowledgment

This work is partially supported by ARC grant LP130100164 and NSF CDI award OIA-1028162.

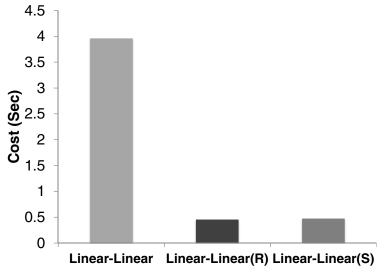


Fig. 13: DIAB: Impact of View Refinement and Skipping Approximations on Cost

References

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

[2] www.basketball-reference.com.

[3] E. Bertini. Quality metrics in high-dimensional data visualization: An overview and systematization. *IEEE Trans. Vis. Comput. Graph.*, 17(12):2203–2212, 2011.

[4] N. Bruno et al. Generating queries with cardinality constraints for dbms testing. *IEEE Trans. Knowl. Data Eng.*, 18(12):1721–1725, 2006.

[5] G. Cormode et al. Histograms and wavelets on probabilistic data. *IEEE Trans. Knowl. Data Eng.*, 22(8):1142–1157, 2010.

[6] G. Cormode et al. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases*, 4(1-3):1–294, 2012.

[7] Q. Cui et al. Measuring data abstraction quality in multiresolution visualizations. *IEEE Trans. Vis. Comput. Graph.*, 12(5):709–716, 2006.

[8] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 66(4):614–656, 2003.

[9] S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of data exploration techniques. In *SIGMOD*, pages 277–281, 2015.

[10] I. F. Ilyas et al. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4), 2008.

[11] Y. E. Ioannidis. The history of histograms (abridged). In *VLDB*, 2003.

[12] H. V. Jagadish et al. Optimal histograms with quality guarantees. In *VLDB*, 1998.

[13] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer. Profiler: integrated statistical analysis and visualization for data quality assessment. In *AVI*, pages 547–554, 2012.

[14] A. Key, B. Howe, D. Perry, and C. R. Aragon. Vizdeck: self-organizing dashboards for visual analytics. In *SIGMOD*, pages 681–684, 2012.

[15] A. Kim et al. Rapid sampling for visualizations with ordering guarantees. *PVLDB*, 8(5):521–532, 2015.

[16] Z. Liu, B. Jiang, and J. Heer. *imMens*: Real-time visual querying of big data. *Comput. Graph. Forum*, 32(3):421–430, 2013.

[17] A. Marian et al. Evaluating top-*k* queries over web-accessible databases. *ACM Trans. Database Syst.*, 29(2):319–362, 2004.

[18] Y. Sakurai, S. Papadimitriou, and C. Faloutsos. BRAID: stream mining through group lag correlations. In *SIGMOD*, pages 599–610, 2005.

[19] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans. Vis. Comput. Graph.*, 8(1):52–65, 2002.

[20] S. Subramaniam et al. Online outlier detection in sensor data using non-parametric models. In *VLDB*, pages 187–198, 2006.

[21] M. Vartak, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: automatically generating query visualizations. *PVLDB*, 7(13):1581–1584, 2014.

[22] M. Vartak, S. Rahman, S. Madden, A. G. Parameswaran, and N. Polyzotis. SEEDB: efficient data-driven visualization recommendations to support visual analytics. *PVLDB*, 8(13):2182–2193, 2015.

[23] E. Wu, L. Battle, and S. R. Madden. The case for data visualization management systems. *PVLDB*, 7(10):903–906, 2014.

Authorized licensed use limited to: University of Pittsburgh Library System. Downloaded on November 11,2025 at 21:29:49 UTC from IEEE Xplore. Restrictions apply.