IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 5, 2021, accepted April 7, 2021, date of publication May 13, 2021, date of current version June 1, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3079932

Efficient Query Refinement for View Recommendation in Visual Data Exploration

MOHAMED A. SHARAF AND HUMAIRA EHSAN
Department of Computer Science and Software Engineering, College of Information Technology, United Arab Emirates University, Abu Dhabi, United Arab Emirates

Corresponding author: Mohamed A. Sharaf (msharaf@uaeu.ac.ae)

This work was supported in part by the United Arab Emirates University under Grant G00003352.

ABSTRACT The need for efficient and effective data exploration has resulted in several solutions that automatically recommend interesting visualizations. The main idea underlying those solutions is to automatically generate all possible views of data, and recommend the top-k interesting views. However, those solutions assume that the analyst is able to formulate a well-defined query that selects a subset of data, which contains insights. Meanwhile, in reality, it is typically a challenging task to pose an exploratory query, which can immediately reveal some insights. To address that challenge, in this work we propose utilizing query refinement as one technique that allows to automatically adjust the analyst's input query to discover such valuable insights. However, a naive query refinement, in addition to generating a prohibitively large search space, also raises other problems such as deviating from the user's preference and recommending statistically insignificant views. In this paper, we address those problems and propose a novel suit of schemes, which efficiently navigate the refined queries search space to recommend the top-k insights that meet all of the analyst's pre-specified criteria.

INDEX TERMS Visual data exploration, data visualization, query refinement, view recommendation.

I. INTRODUCTION

Visual data exploration is becoming a key component in a widely diverse set of discovery-oriented applications in healthcare monitoring, manufacturing, financial analysis, education, and transportation planning, just to name a few [1], [2]. Visual data exploration typically involves an analyst going through the following steps: 1) selecting a subset of data, 2) generating different visualizations of that subset of data, and 3) sifting through those visualizations for the ones which reveal interesting insights. Based on the outcome of the last step, the analyst might have to refine their initial selection of data so that the new subset would show more interesting insights. This is clearly an iterative and time-consuming process, in which each selection of data (i.e., exploratory input query) is a springboard to the next one.

Motivated by the need for an efficient and effective visual data exploration process, several solutions have been proposed towards automatically finding and recommending interesting data visualizations (i.e., steps 2 and 3 above) (e.g., [3]–[8]). The main idea underlying those solutions is to automatically generate all possible views of the explored data, and recommend the top-k *interesting* views, where the 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 [3], [6], [7], [9]. In particular, the deviation-based metric measures the distance between the probability distribution of a visualization over the analyzed dataset (i.e., *target view*) and that same visualization when generated from a comparison dataset (i.e., *comparison view*), where the comparison dataset is typically the entire database. The underlying premise is that a visualizations that results in a higher deviation is expected to reveal insights that are very particular to the analyzed dataset.

Existing solutions have been shown to be effective in recommending interesting views under the assumption that the analyst is ''precise'' in selecting their analyzed data. That is, the analyst is able to formulate a well-defined exploratory query, which selects a subset of data that contains interesting insights to be revealed by the recommended visualizations. However, such assumption is clearly impractical and extremely limits the applicability of those solutions. In reality, it is typically a challenging task for an analyst to select a subset of data that has the potential of revealing interesting

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaoou Li.

insights. Hence, it is a continuous process of trial and error, in which the analyst keeps *refining* their selection of data manually and iteratively until some interesting insights are revealed. Therefore, in this work we argue that, in addition to the existing solutions for automatically recommending interesting views, there is an equal need for solutions that can also automatically select subsets of data that would potentially provide such interesting views. Hence, our goal in this work is not only to recommend interesting views, but also to recommend exploratory queries that lead to such views. To further illustrate the need for such solution, consider the following example.

Example 1: Consider an analyst wants to explore and find interesting insights in the U.S. Census income dataset [10], which is stored in table C. Her intuition is that analyzing the subset of data of those who have achieved a high level of education might reveal some interesting insights. Therefore, she selects that particular subset in which everyone has completed their 12^{th} year of education (i.e., graduated high school) via the following query:

Q: `SELECT * FROM C WHERE education` $\geq$ `12`

To find the top-k visualizations, she might use one of the existing approaches (e.g., [3], [7]), in which all the target and comparison aggregate views are generated and their deviation is computed by using a distance function (e.g., Euclidean distance). Figure 1 shows the top-k visualization recommended by such approaches. Specifically, the figure shows that among all the attributes in the Census dataset, the recommended most interesting visualization is based on plotting the probability distribution of the `Hours per week` attribute of the dataset. That is, a histogram-like distribution of the number of hours worked per week for those who graduated high school (i.e., education $\geq$ 12) vs. the population. Such visualization is equivalent to plotting the probability distributions of the target view V_t and the comparison View V_c, which are expressed in SQL in Figure 1. Hence, the deviation value shown in Figure 1 is the Euclidean distance between the probability distribution of V_t and V_c. However, by carefully examining Figure 1, it is clear that there is not much difference between those who graduated high school and the population with respect to the `Hours per week` dimension. That is, the target and comparison views are almost the same, which is also reflected by the low-deviation value of 0.0459. Despite that, such visualization would still be recommended by existing approaches because it achieves the maximum deviation among all the views generated over the data subset selected by query Q, even though that maximum deviation value is inherently low.

The previous example illustrates a clear need for a query refinement solution that is able to automatically modify the analyst's initial input query and recommend a new query, which selects a subset of data that includes interesting insights. Those hidden insights are then easily revealed using existing solutions that are able to recommend interesting visualizations. To that end, one straightforward and simple approach would involve generating all the possible subsets of data by automatically refining the predicates of the input query. In our example above, that would be equivalent to generating all refinements of the predicate `WHERE education` $\leq$ `12`. Consequently, for each subset of data selected by each query refinement, generate all possible aggregate views (i.e., visualizations). In addition to the obvious challenge of a prohibitively large search space of query refinements, that naive approach would also lead to visualizations that might appear to be visually interesting but they are irrelevant from the analyst's perspective, which is illustrated in the following example.

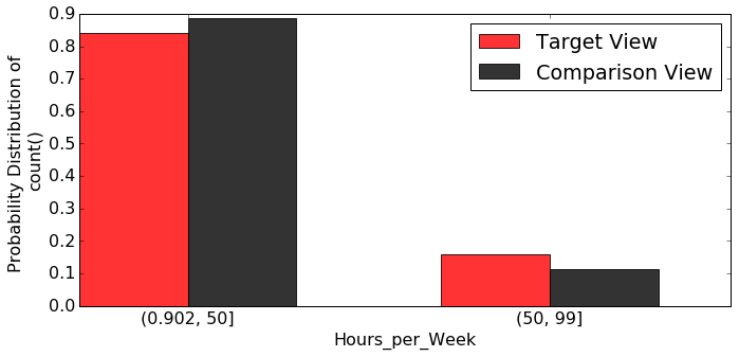
Example 2: Now assume that the naive approach described above is applied to Example 1, such that all possible refinements of Q are generated so that to find and recommend visualizations that are more interesting than those in Figure 1. Particularly, after generating all those possible refined queries with all the possible values for the predicate on the `education` attribute, the recommended top visualization is shown in Figure 2. That visualization is recommended based on the following refined query:

Q_1: `SELECT * FROM C WHERE education` $\leq$ `1`

As Figure 2 shows, that visualization is based on plotting the probability distribution of the `occupation` attribute for those who never went to school (i.e., `education` $\leq$ `1`) vs. the population. Clearly, Figure 2 is visually interesting as the probability distribution of the target view is significantly different from the comparison view (i.e., deviation = 0.2614). However, there are two issues with that refinement, and in turn the recommended visualization:

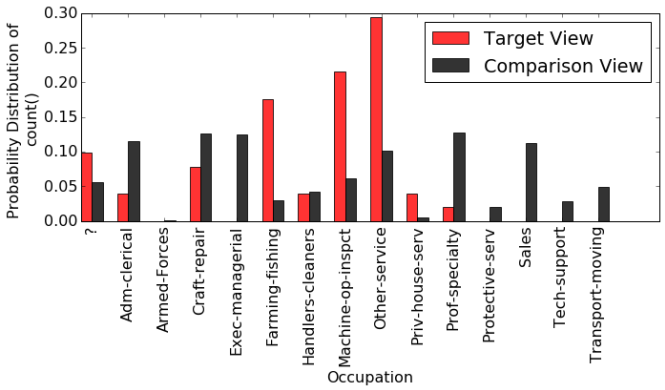
1) *Similarity-oblivious*: a blind automated refinement that is oblivious to the analyst's preferences might result in a refined query that is significantly dissimilar from the input query. For instance, in this example the analyst's intention is to analyze the subset of data for those who completed high school (i.e., `education` $\geq$ `12`), whereas the refined query selects for the analysis those who never went to school (i.e., `education` $\leq$ `1`).
2) *Statistical insignificance*: the subset selected by the refined query can be too small and as a result the target views generated from that subset will miss a number of values for the dimension attribute. This leads to views with high deviation values but statistically insignificant. As shown in Figure 2, the target view is missing a number of values for the `occupation` attribute, which indicates that the subset selected by the refined query Q_1 is possibly too small for analysis, and in turn statistically insignificant.

The two issues mentioned above highlight the need for automatic refinement solutions that are guided by the user's preference and statistical significance, which is the focus of this work. In particular, we propose a novel scheme for automated query refinement for view recommendation in visual data exploration, called **QuRVe**. Before discussing the details of QuRVe in the next sections, we illustrate its benefits using the following example:



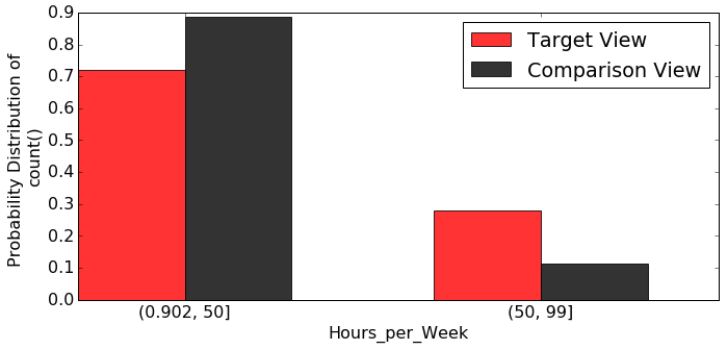
Deviation = 0.0459
V_t: SELECT HoursPerWeek, COUNT(*) FROM C
WHERE education ≥ 12
GROUP BY HoursPerWeek
V_c: SELECT HoursPerWeek, COUNT(*) FROM C
GROUP BY HoursPerWeek

FIGURE 1. View on input query Q.



Deviation = 0.2614
V_t: SELECT Occupation COUNT(*) FROM C
WHERE education ≤ 1
GROUP BY Occupation
V_c: SELECT Occupation COUNT(*) FROM C
GROUP BY Occupation

FIGURE 2. View on refined query Q_1.



Deviation = 0.169
V_t: SELECT Hours_Per_Week, COUNT(*) FROM C
WHERE education ≥ 16
GROUP BY Hours_Per_Week
V_c: SELECT Hours_Per_Week COUNT(*) FROM C
GROUP BY Hours_Per_Week

FIGURE 3. View on refined query Q_2.

Example 3: Figure 3 shows the top view recommended by QuRVe based on the input query Q provided in Example 1. That view is generated based on automatically refining Q into the new modified and statistically significant query Q_2, which is specified as: Q_2 : `SELECT * FROM C WHERE education ≥ 16`. Notice that Q_2 is clearly more similar to the input query Q than the previously refined query Q_1 from Example 2. Particularly, instead of selecting the data for those who completed high school (i.e., `education ≥ 12`), the refined query Q_2 selects those who completed a college degree (i.e., `education ≥ 16`). Equally important, the recommended view based on the refined Q_2 shows a uniquely interesting insight. Specifically, as Figure 3 shows, highly educated people tend to work more hours than the rest of the population. More precisely, only 13% of the population work more than 50 hours a week, whereas for those who have completed college that percentage goes up to 30%.

Based on the previous example, our key goal in designing QuRVe is to recommend interesting visualizations, while at

the same time achieving high effectiveness and efficiency. To ensure that desired effectiveness, we formulate the problem of refining query for recommending top-k aggregate visualization as a multi-objective optimization problem. Particularly, given an input query Q, the optimization objective is to find those top-k interesting visualizations from all possible refinements of the input query according to a similarity-aware utility function, subject to predefined constraints on statistical significance. Clearly, such formulation is challenged by the large number of possible refinements and the corresponding visualizations generated per refinement.

After the QuRVe framework was first introduced in the second author's PhD thesis [8], our later work in [11] showed that QuRVe is able to efficiently reduce the prohibitively large search space of possible views by utilizing some of the salient characteristics of our multi-objective optimization problem described above. In this work, we expand on our basic QuRVe, and propose two new novel schemes that are able to further reduce the search space of refined queries, and in turn minimize the query execution cost incurred in the process of recommending those interesting aggregate visualizations. Particularly, the key idea underlying the QuRVe family of schemes is to calculate an upper bound on the maximum possible utility achieved by each view without actually executing the aggregate query that generates that view. Then only those promising views with expected high-utility are processed and the top-k are recommended. This allows QuRVe to prune a large number of unnecessary views, and in turn reduces the overall processing time for recommending the top-k views. However, our original QuRVe [11] utilizes a theoretical loose bound when estimating that maximum possible utility provided by a view. That theoretical bound is oblivious to the characteristics of the analyzed data, and in turn tends to overestimate the utility provided by each view. Consequently, it limits QuRVe's power in pruning some unnecessary low-utility views.

To address that limitation and achieve higher efficiency in view recommendation, in this work we propose the uQuRVe and pQuRVe schemes. Both schemes utilize the characteristics of the analyzed data to estimate a tight upper bound on the utility of a view. Hence, they allow pruning more unnecessary views, and save the costs for processing those eliminated views. Furthermore, pQuRVe controls the order of navigating the views search space so that high-utility views are visited and processed first, and consequently finding the top-k views rather early in the search process.

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

- We formulate the problem of query refinement for recommending visualizations and introduce a novel hybrid multi-objective utility function, which guides the query refinement and view recommendation process.
- We propose the QuRVe, uQuRVe and pQuRVe schemes, which introduce novel search algorithms that are particularly optimized to leverage the specific features of the problem for pruning the search space.
- We conduct extensive experimental evaluation on real datasets, which illustrate the benefits achieved by the QuRVe family of schemes.

Roadmap: In Section II, we describe the background for our problem. We formally describe our multi-objective utility function and problem statement in Section III. Our proposed search algorithms are discussed in Section IV. We present our testbed and experiments on real data in Sections V and VI. The related work is discussed in Section VII, and finally conclusion in Section VIII.

II. PRELIMINARIES

In this section, we first describe the basics of view recommendation, followed by automatic query refinement and statistical significance for view recommendation. All symbols are summarized in Table 1.

TABLE 1. Summary of symbols.

Symbol	Description
D	Database
D_S	Subset of data selected by Q
Q	Input query
T	Predicates in input query
Q_j	A refined query
$\mathbb{Q}$	Set of refined queries
$\mathbb{A}$	Set of dimension attribute
$\mathbb{M}$	Set of measure attribute
$\mathbb{F}$	Set of aggregate function
$\mathbb{P}$	Set of attributes for predicates
$V_i(D_S)$	i^{th} Aggregate view on D_S
$V_{i,Qj}$	i^{th} Aggregate view on refined query Q_j
N	Total number of views
$D(V_{i,Q_j})$	Deviation of a view V_{i,Q_j}
$S(Q, Q_j)$	Similarity between Q and Q_j
D_M	Theoretically maximum deviation of any view
D_u	Upper bound on deviation of any views

A. VIEW RECOMMENDATION

Similar to recent work on visual data exploration (e.g., [3], [7]), we assume a multi-dimensional dataset $D(\mathbb{A}, \mathbb{M})$, where $\mathbb{A}$ is the set of dimension attributes, $\mathbb{M}$ is the set of measure attributes. Further, $\mathbb{F}$ is the set of possible aggregate functions over the measure attributes $\mathbb{M}$. In a typical visual data exploration session the user chooses a subset D_S of the dataset D by issuing an input query Q. For instance, consider the query Q: `SELECT * FROM D WHERE T;` In Q, T specifies a combination of predicates, which selects D_S for visual analysis (e.g., `education` $\geq$ `12` in Example 1). A visual representation of Q is basically the process of generating an aggregate view V_i of its result (i.e., D_S), which is then plotted using some visualization methods such as bar charts, scatter plots, etc. Therefore, an aggregate view V_i over D_S is represented by a tuple (A, M, F, b) where $A \in \mathbb{A}$, $M \in \mathbb{M}$, $F \in \mathbb{F}$ and b is the number of bins in case A is numeric. That is, D_S is grouped by dimension attribute A and aggregated by function F on measure attribute M. For

instance, the tuple `(Hours per Week, *,COUNT,2)` represents the aggregate view shown in Figure 1.

Manually finding interesting and insightful views of data is a time-consuming task. Towards automated visual data exploration, recent approaches have been proposed for recommending interesting visualizations based on deviation based metric (e.g., [3], [7]). In particular, that metric measures the deviation between the aggregate view V_i generated from the subset data D_S vs. that generated from the entire database D, where $V_i(D_S)$ is denoted as *target* view, whereas $V_i(D)$ is denoted as *comparison* view. To ensure that all views have the same scale, each target view $V_i(D_S)$ and comparison view $V_i(D)$ is normalized into a *probability distribution* $P[V_i(D_S)]$ and $P[V_i(D)]$ and it is bounded by the maximum deviation value D_M. Accordingly, the deviation $D(V_i)$, provided by a view V_i, is defined as the normalized distance between those two probability distributions.

$$D(V_i) = \frac{dist(P[V_i(D_S)], P[V_i(D)])}{D_M} \quad (1)$$

Then, 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*) [3], [7], [9], [12]. 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.

As explained earlier, the input exploratory query Q is the cornerstone for recommending interesting views. However, formulating an input query that results in interesting views is a non-trivial task. Therefore, in this work we propose to automatically refine that input query to select subsets of data that reveal interesting insights. In the next section we discuss the preliminaries of query refinement.

B. QUERY REFINEMENT

Automatic query refinement is a widely used technique for DBMS testing, information retrieval and data exploration. In a nutshell, in this technique the user provides an initial query and then it is progressively refined to meet a particular objective [13]–[16]. In the context of data exploration and aggregate queries, query refinement has been used to automatically recommend queries satisfying cardinality and aggregate constraints [13], [14], explaining outliers [15] and answering why not questions [16]. In this work, we propose to automatically refine an input exploratory query for the objective of view recommendation. Particularly, as mentioned in Section II-A, the user provides an input query Q, in which T specifies a conjunction of predicates. Then Q is progressively refined by automatically enumerating all combinations of predicates for the objective of generating interesting views.

Particularly, we consider queries having selection predicates with range $(<, \leq, >, \geq)$ operators. These predicates are defined on a set of numeric dimension attributes denoted as $\mathbb{P}$. The number of predicates is p, such that $|\mathbb{P}| = p$. Each of those range predicates is in the form $l_i \leq P_i \leq u_i$ where $P_i \in \mathbb{P}$ and l_i and u_i are the lower and upper limits of query Q along predicate P_i. The domain of predicate P_i is limited by a Lower bound L_i and upper bound U_i.

A refined query Q_j for an input query Q is generated by modifying the lower and/or upper limits for some of the predicates in Q. That is, for a predicate $l_i \leq P_i \leq u_i$ in query Q, a refined predicate in Q_j takes the form $l'_i \leq P_i \leq u'_i$. Notice that a predicate that is not included in T would be equivalent to $L_i \leq P_i \leq U_i$.

Similar to [13], [17], we convert a range predicate into two single-sided predicates. Therefore, $l_i \leq P_i \leq u_i$ is converted to two predicates: $P_i \leq u_i \bigwedge -P_i \leq -l_i$. This allows refinement of one or both sides of the range predicates and this results in the total number of single sided predicates to be $2p$. Consider the following example to clearly understand how to convert a range to two single-sided predicates.

Example: Consider the query Q in Example 1, in Q only $l_i = 12$ is defined explicitly, however $u_i = U_i = 16$ is automatically added and Q is completely defined as, Q: `SELECT * FROM C WHERE 12 ≤ education ≤ 16`. After converting to single sided predicated it would become Q: `SELECT * FROM C WHERE education ≤ 16 AND -education ≤ -12`

Finally, query refinement can be in one of the two directions: i) *contracting* i.e., decreasing the value of the predicate and ii) *expanding* i.e., increasing the value of the predicate. The set of all of the refined queries is denoted as $\mathbb{Q}$. Clearly, the number of all possible refinements is exponential in p and forms a combinatorial search space. For instance, if predicate P_i is discrete and is refined in steps of 1.0, then the number of all possible refinements of P_i is $n_i = \frac{a_i(a_i+1)}{2}$, where $a_i = U_i - L_i$. For a total of p such predicates, the combinations of all possible refinements is: $n_1 \times n_2 \times \ldots\ldots n_p \approx n^p$. In other words, the size of the set $\mathbb{Q}$ is approximately n^p (i.e., $|\mathbb{Q}| \approx n^p$).

A refined query Q_j is obtained by changing one or more predicates $P_i \in T$ to P'_i, which naturally makes the refined query Q_j different from the input query Q. However, a refined query that is significantly dissimilar from its counterpart input query would result in loss of user preference and might be deemed irrelevant to the analysis. Hence, to quantify the change made to transform Q into the refined query Q_j, we define a similarity measure $S(Q, Q_j)$ in terms of the distance between Q_j and Q (i.e., $s(Q, Q_j)$).

$$S(Q, Q_j) = 1 - s(Q, Q_j) \quad (2)$$

While the exact specification of $s(Q, Q_j)$ is deferred to Section III, it is worth pointing out the impact of query refinement on the deviation computation defined in Eq. 1. Particularly, when utilizing refinement, a view V_i can be either generated from the input query, or a refined one. To associate each view with its underlying query, we denote a view as V_{i,Q_j} to specify the i^{th} view generated over the result of query Q_j. Accordingly, Eq. 1 is modified to define the

deviation $D(V_{i,Q_j})$ of a view V_{i,Q_j}, as:

$$D(V_{i,Q_j}) = \frac{dist(P[V_i(D_{Q_j})], P[V_i(D)])}{D_M} \quad (3)$$

C. HYPOTHESIS TESTING

In visual data exploration, it is often the case that an observed high-deviation is actually statistically insignificant. This problem leads to misleading ranking of such views, and in turn inaccurate recommendations [18]–[20]. For instance, in our recent work on the *MuVE* scheme [7], [12], we made the following observations:

1) Some of the recommended top-k target views have very few underlying tuples, which leads to higher deviation values. Consequently, such views receive higher rank despite of the lack of real insight, for instance, the target view shown in Figure 2 of Example 2.
2) Often the data selected by the exploratory query result in only low-deviation views. Consequently, the top-k recommend views will exhibit low-deviation, as shown in Example 1. However, such top-k recommendations are clearly statistically insignificant.

To determine whether the observed difference is statistically significant, we employ the widely used approach hypothesis testing. Hypothesis testing determines if there is enough evidence for inferring that a difference exists between two compared samples or between a sample and population. A difference is called statistically significant if it is unlikely to have occurred by chance [18]. Hypothesis testing involves testing a null hypothesis by comparing it with an alternate hypothesis. The hypothesis to be tested is called the *null hypothesis*, denoted as H_0. The null hypothesis states that there is no difference between the population and the sample data. The null hypothesis is tested against an *alternate hypothesis*, denoted as H_1, which is what we have observed in the sample data. For instance, in Figure 1 of Example 1, the hypothesis is that *"high school graduates work different number of hours per week (Hours worked is divided into two categories) as compared to the population"*, and this becomes H_1. The corresponding H_0 is that no such difference exits. Likewise, each possible view V_i from each refined query become a H_1, which is to be tested for significance before recommendation.

Depending on the nature of the statistical test and the underlying hypothesis, different null hypothesis statistical tests have been developed, e.g., chi-square test for categorical dimension attributes. Furthermore, after stating H_0 and H_1, the chosen statistical test returns a *p-value*. The p-value is the probability of obtaining a statistic at least as extreme as the one that was actually observed, given H_0 is true. Specifically, the p-value is compared against a priori chosen *significance level* α, where the conventionally used significance level is 0.05. Hence, if $pvalue(V_i) \leq \alpha$, then H_0 must be rejected, which means the V_i is statistically significant. Clearly, due to the nature of the statistical test involved, the acceptance or rejection of H_0 can never be free of error. If the test incorrectly rejects or accepts H_0, then an error has occurred. Hypothesis testing can incur the following two types of error: 1) If H_0 is rejected, while it was true, it is called *Type-I error* and 2) If H_0 is accepted, while H_1 was true, it is called *Type-II error*. Type-II error is critical in our case because we want to avoid rejecting views that might be interesting. The probability of Type-II error is specified by a parameter β, which normally has a value $0.10 - 0.20$. An alternate term is power, which is the probability of rejecting a false H_0, therefore, $power = 1 - \beta$. A priori power analysis is employed to determine the minimum sample size that is necessary to obtain the required power. By setting an effect size (ω), significance level (α), and power level (β), the sample size to meet specification can be determined [21].

Putting it together, blindly applying query refinement in search for queries that generate high-deviation views can often lead to the paradox of selecting queries with very few underlying tuples. Accordingly, the target views generated from those queries typically have missing groups, and in turn misleadingly exhibit high-deviation from the comparison view.

This leads to false discoveries as the high deviation is because of insufficient sample size, and not due to the different distribution of target view from comparison view. Therefore, we perform power analysis, estimate the minimum sample size required to achieve the specified power, and employ it as a constraint in our problem definition. Consequently, only the refined subsets that satisfy the minimum sample constraint participate in our search of top-k views.

III. PROBLEM DEFINITION

In this section, we formally define the problem of query refinement for 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 and all its corresponding refined queries $Q_j \in \mathbb{Q}$, according to some utility function. When the recommended visualizations are only based on query Q, such goal simply boils down to recommending the top-k interesting views based on the deviation metric, as described in Section II-A. However, that simple notion of utility falls short in capturing the impact of refinement on the input query. In particular, automatic refinement introduces additional factors that impact the level of interestingness, and in turn the utility of the recommended views. Accordingly, in our proposed scheme, we employ a weighted multi-objective utility function and constraints to integrate such factors. In particular, for each view V_{i,Q_j}, we evaluate the following components:

1) *Interestingness:* is the ability of view V_{i,Q_j} to reveal some insights about the data, which is measured using the deviation-based metric $D(V_{i,Q_j})$ (Eq. 3).
2) *Similarity:* is the similarity between the input query Q, and the refined query Q_j underlying the view V_{i,Q_j}, which is measured as $S(Q, Q_j)$ (Eq. 2).
3) *Statistical Significance:* is the ability of the refined query Q_j and the view V_{i,Q_j} to generate a statistically

significant result, which is captured by checking that the size of the subset selected by Q_j satisfies the constraint $power(Q_j)$, and the significance of the view V_{i,Q_j} satisfies the constraint $pvalue(V_{i,Q_j})$.

To capture the factors and constraints mentioned above, we employ a weighted multi-objective utility function, which is defined as follows:

$$U(V_{i,Q_j}) = \alpha_S \times S(Q, Q_j) + \alpha_D \times D(V_{i,Q_j}) \quad (4)$$

where $S(Q, Q_j)$ is the similarity between input query Q and refined query Q_j of the view V_{i,Q_j}, and $D(V_{i,Q_j})$ is the normalized deviation of view V_{i,Q_j} from the overall data. Parameters α_S and α_D specify the weights assigned to each objective in our hybrid utility function, such that $\alpha_S + \alpha_D = 1$. Those weights can be user-defined so that to reflect the user's preference between interestingness and similarity. 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 under specified constraints.

Definition Query Refinement for View Recommendation: Given a user-specified query Q on a database D, a multi-objective utility function U, a significance level α, statistical power $1 - \beta$ and a positive integer k. Find k aggregate views that have the highest utility values, from all of the refined queries $Q_j \in \mathbb{Q}$ such that $pvalue(V_{i,Q_j}) \leq \alpha$ and $power(Q_j) > 1 - \beta$.

In short, the premise is that a view is of high utility for the user, if it satisfies the specified constraints, shows high-deviation, and is based on a refined query that is highly similar to the user specified query. To estimate the cost incurred in solving the problem defined above, note that for each refined query $Q_j \in \mathbb{Q}$, the number of aggregate queries posed to the database equals to the number of aggregate views generated i.e., $2 \times |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$. Furthermore, as discussed in the previous section, the number of refined queries is exponential to the number of predicates p. Therefore, with query refinement, the total number of candidate views N is: $N \approx n^p \times 2 \times |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$. Clearly, this is a very large search space and requires an effective navigation with minimum cost.

Before introducing our search schemes in the next section, we first provide the detailed definition of the similarity component of our objective function mentioned above (i.e., $S(Q, Q_j)$). Particularly, to fully define the similarity component of our utility function, we revisit Eq. 2 which quantifies the distance between Q and Q_j. In the literature, a number of methods have been proposed to measure the distance between two range queries [16], [22], [23]. Similar to [14], [17], we calculate the distance in terms of absolute change in predicate values (Eq. 5). This method provides a reasonable approximation of the change in data selected by the refined query at a negligible cost. Additionally, we normalize it by predicate bounds to accommodate the different scales of various predicates. Recall that Eq 2 quantifies the change in an input query Q to get to a refined query Q_j, in terms of the distance between Q and Q_j.

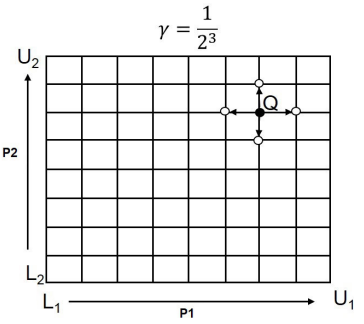


FIGURE 4. Query space $\mathbb{Q}$.

$$s(Q, Q_j) = \frac{1}{p} \sum_{i=1}^{p} \frac{|l_i^{Q_j} - l_i^{Q}| + |u_i^{Q_j} - u_i^{Q}|}{2|U_i - L_i|} \quad (5)$$

Example: Consider Q and Q_1 from Examples 1 and 2 respectively. Note that the range of predicate education is [1,16]. Hence, the distance between Q and Q_1 is: $s(Q, Q_1) = \frac{|1-12|+|1-16|}{2|16-1|} = 0.86$ In other words this mean there is 14% similarity between Q and Q_1. Similarly, Consider Q and Q_2 in Example 3, note that in this case Q and Q_2 have 87% similarity.

IV. SEARCH SCHEMES

For an input query Q, each possible query refinement of Q can be represented as a point in a p-dimensional space, where $|\mathbb{P}| = p$ (please see Section II-B for more details). Clearly, one of the points in that space is the input query Q itself, and the remaining points belong to the set of refined queries $\mathbb{Q}$. Our high-level goal is to: 1) generate the set $\mathbb{Q}$, 2) compute the utility of all the aggregate views generated from each query in $\mathbb{Q}$, and 3) recommend the top-k views after ranking them based on their achieved utility. To that end, clearly the large size of $\mathbb{Q}$ and the corresponding aggregate views, together with the complexity of evaluating the statistical significance and utility function of each view, makes the problem highly challenging. Hence, in this section, we put forward various search strategies for finding the top-k views for recommendation.

A. THE LINEAR SCHEME

Clearly, a naive way to identify the top-k objects is to use a *linear* scheme, which scores all objects based on a scoring function, sort, and return the top-k objects. Accordingly, this Linear scheme is basically an exhaustive and brute force strategy, in which views from all refined queries are generated and ranked according to their utility. As we consider predicates on continuous dimensions, infinite possible values can be assigned to predicates in those refined queries. Therefore, each dimension is discretized with a user specified parameter γ, which divides the range of dimension attribute into $1/\gamma$ equi-width intervals. That discretization results in a grid, where a grid for a two-dimensional space is shown in Figure 4.

In this linear scheme, irrespective of Q, iteratively all refined queries are generated using all combinations of Predicates $P_1, P_2 \ldots P_p$. For instance, in Figure 4, each intersection point on the grid represent a refined query at $\gamma = \frac{1}{2^3}$ having dimensions P_1 and P_2,.

For each query $Q_j \in \mathbb{Q}$, to check the constraint $power(Q_j) < 1 - \beta$, a function *powerTest(Q_j, ω, β, α)* is defined, which returns true value if the constraint is satisfied, else it returns false. The cost of checking this constraint is one database probe, where a `COUNT` query with predicates of Q_j is executed to get the sample size of Q_j. Moreover, for the queries that satisfy the statistical power constraint, all views are generated. Then for each view V_{i,Q_j} the constraint $pValue(V_{i,Q_j}) < \alpha$ is checked. Specifically, for this purpose, another function *significanceTest(V_{i,Q_j}, α)* is defined, which returns a true value if p-value$< \alpha$. Consequently, for each view V_{i,Q_j} that satisfies the constraint, its utility value $U(V_{i,Q_j})$ is computed, and finally the top-k views are returned.

B. THE QuRVe SCHEME

Clearly, the linear search scheme, described above, visits every possible view, therefore, it is very expensive in terms of execution time. In this section, we present the QuRVe scheme, which reduces cost by pruning a large number of views.

Notice that our problem of finding top-k views is similar to the problem of top-k query processing, which is extensively studied in various settings [24]. Generally in these settings objects are evaluated by multiple objectives that contribute to the overall score of each object. A scoring/utility function is therefore usually defined as an aggregation over partial scores of defined objectives. In terms of efficiency, the best performing techniques for various top-k problem settings are based on the threshold algorithm (TA) [24], [25]. TA generates sorted lists of objects on partial scores for every objective, visits the lists in round robin fashion and merges objects from different lists to compute the aggregated scores. Typically, it *early terminates* the search as soon as it has the *top-k* objects (i.e., long before reaching the end of the lists).

In our settings, we have a similar configuration. That is, we have two partial scores of a view V_{i,Q_j}, namely: 1) Similarity score $S(Q, Q_j)$, and 2) Deviation score $D(V_{i,Q_j})$. Those scores are maintained in two lists: S_{list} and D_{list}. Conversely, we also have some key differences: 1) for any view V_{i,Q_j} the values of $S(Q, Q_j)$ and $D(V_{i,Q_j})$ are not physically stored and are computed on demand, 2) calculating $D(V_{i,Q_j})$ for a view is an expensive operation, and 3) the size of the view search space is prohibitively large and potentially infinite.

Obviously, a forthright implementation of TA is infeasible to our problem due to the limitations mentioned before. However, recall that the similarity objective $S(Q, Q_j)$ is the comparison of predicates of Q_j with Q and involves no database probes. Hence, a sorted list S_{list} can be easily generated at a negligible cost. However, populating the D_{list} in a similar fashion is not possible, as it involves expensive database probes. Therefore, to minimize the number of probes and efficiently populate D_{list}, the Sorted-Random (SR) model of the TA algorithm [24] is employed. In the SR model the sorted list (S) provides initial list of candidates and the random list (R) is probed only when required. Accordingly, QuRVe provides S_{list} as the initial list of candidate views, by incrementally generating refined queries in decreasing order of similarity and populating the S_{list}. The views in S_{list} have their partial scores, the final scores are only calculated for the views for which the D_{list} is also accessed. To limit the number of those views, QuRVe maintains the following variables:

1) U_{Unseen}: Stores the maximum possible utility of the views that are not probed yet.
2) U_{Seen}: Stores the k^{th} highest utility of a view seen so far.

Specifically, to calculate U_{Unseen}, the upper bound on deviation is used. Particularly, consider V_{i,Q_j} as the next view in S_{list} and let the upper bound on its deviation be $D_u(V_{i,Q_j})$. Moreover, let the normalized upper bound on deviation from all views be D_u then $D_u = Max[D_u(V_{i,Q_j})]$. Consequently, $U_{Unseen} = \alpha_S \times S(Q, Q_j) + (1 - \alpha_S) \times D_u$. To set and normalize the value for D_u, we notice that the maximum possible deviation is simply $\sqrt{2}$ (for the sake of brevity, the derivations needed to obtain that value are detailed in the next section). Given that maximum value, and after normalization, then theoretically $D_u = Max[D_u(V_{i,Q_j})] = 1$.

In detail, QuRVe starts with initializations as: (i) there are no views generated yet, therefore $U_{Seen} = 0$, (ii) $U_{Unseen} = D_u$, and (iii) Q has the highest similarity i.e., $S(Q, Q) = 1$, therefore, Q is added to Q_{list} as the first member. Then, the power of the currently under consideration query Q_j is checked by the function *powerTest(Q_j, ω, β, α)*. Next, the corresponding views are generated by the function *generateViews(Q_j)* and the statistical significance test is performed on each view by the function *significanceTest(V_{i,Q_j}, α)*. The Utility of the views that pass the test is computed. Accordingly, the list *topk* is updated. The utility of k^{th} highest view is copied into U_{Seen}, to maintain the bound on the seen utility values. This completes processing the currently under consideration query Q_j. Later, the next set of neighboring queries are generated.

In the next iteration, another query Q_j is taken from the Q_{list} in order of the similarity objective value and accordingly the value of U_{Unseen} is updated. The iterations continue, until either there are no more queries to process, or the utility of the remaining queries will be less than the already seen utility (i.e., $U_{Unseen} > U_{Seen}$ is false). If QuRVe terminates because of the first condition that means its cost is the same as Linear search, as the early termination did not get a chance to step in. However, often QuRVe terminates because of the second condition (i.e., $U_{Unseen} > U_{Seen}$ is false) and achieves early termination.

Example: Consider the example shown in Figure 5a, which shows 8 views ($V_1 - V_8$) that are based on 8 queries ($Q_1 - Q_8$), where Q_1 is the original input query, and the rest

V_i	$S(V_i)$	$D(V_i)$	$U(V_i)$	U_{Seen}	U_{Unseen}
V1	1	0.1	0.64	0.64	0.85
V2	0.75	0.1	0.49	0.64	0.85
V3	0.75	0.15	0.51	0.64	0.7
V4	0.5	0.4	0.46	0.64	0.7
V5	0.5	0.34	0.436	0.64	0.7
V6	0.5	0.7	0.58	0.64	0.55
V7	0.25				
V8	0.25				

(a) Example 1: $k = 1, \alpha_S = 0.6, \alpha_D = 0.4$

V_i	$S(V_i)$	$D(V_i)$	$U(V_i)$	U_{Seen}	U_{Unseen}
V1	1	0.1	0.55	0.55	0.875
V2	0.75	0.1	0.425	0.55	0.875
V3	0.75	0.15	0.45	0.55	0.75
V4	0.5	0.4	0.45	0.55	0.75
V5	0.5	0.34	0.42	0.55	0.75
V6	0.5	0.7	0.6	0.6	0.625
V7	0.25	0.5	0.375	0.6	0.625
V8	0.25	0.6	0.425	0.6	

(b) Example 2: $k = 1, \alpha_S = 0.5, \alpha_D = 0.5$

FIGURE 5. The QuRVe scheme.

are refinements of Q_1. Those 8 views are sorted according to their precomputed similarity, as shown in column $S(V_i)$. Naturally, view V_1, which is based on the input query Q_1, appears at the top of the list with $S(V_1) = 1$, followed by the remaining views in descending order of $S(V_i)$. The parameters are set as $k = 1$, $\alpha_S = 0.6$ and $\alpha_D = 0.4$. Additionally, $D(V_i)$ and $U(V_i)$ columns correspond to the deviation and utility values of the view V_i, which are computed on demand. For instance, when view V_1 is probed, its deviation is 0.1. Then its utility is computed as: $0.6 \times 1 + 0.4 \times 0.1 = 0.64$. In linear search, all of the 8 views are probed to compute their deviation and utility values. However, for QuRVe, after probing V_6, $U_{Unseen} = 0.6 \times 0.25 + 0.4 \times 1 = 0.55$ while $U_{Seen} = 0.64$, therefore, the condition $U_{Unseen} > U_{Seen}$ becomes false, hence the search is early terminated and two of the views get pruned.

QuRVe reduces the cost of finding top-k views by pruning unnecessary views. However, it is most efficient with particular settings of input parameters or when the refined query that have the view with the maximum utility is near the input query. For instance, consider Figure 5b, which shows the same example of Figure 5a but with $\alpha_S = \alpha_D = 0.5$. In that particular example, as the deviation has more weight, therefore, the value of U_{Unseen} is bigger, while U_{Seen} is smaller, as compared to Figure 5a and as a result QuRVe fails to prune any views. The performance of QuRVe is restricted by the maximum value of U_{Unseen}. The most efficient performance of QuRVe is expected when U_{Unseen} decreases quickly during search and early termination can be triggered. In the next section, we propose uQuRVe, which is particularly designed to set the utility of not yet seen views (i.e., U_{Unseen}) to a tighter bound instead of the loose upper bound utilized by QuRVe.

C. THE uQuRVe SCHEME

Recall that the fundamental idea underlying our proposed QuRVe scheme is to early terminate the search based on the upper bound of the utility of not yet seen views (i.e., U_{Unseen}). Particularly, U_{Unseen} is computed by using the similarity value of the next view on the list, together with the upper bound on deviation D_u. Hence, if U_{Unseen} is less than the already seen maximum utility (U_{Seen}), then all of the remaining views have no chance of making it up to the top-k. Therefore, QuRVe terminates the search at that point, and those remaining views are pruned. Note that the D_u used by QuRVe to calculate U_{Unseen} is basically a theoretical extreme value and it is oblivious to the analyzed data. Hence, in practice, that upper bound D_u is typically loose as it tends to overestimate the upper bound on deviation between the target and comparison views. Consequently, many chances of pruning unnecessary views are missed.

In our problem setting views are generated corresponding to the input query Q, as well as all its refinements (i.e., $Q_j \in \mathbb{Q}$). In that setting, we observe that while the target views for every Q_j are generated from scratch (i.e., require a separate query execution), the comparison views are only generated once for the input query Q and those same comparison views are reused later for each Q_j. Such observation is the main idea underlying our novel uQuRVe scheme introduced in this section. Particularly, uQuRVe leverages such observation to provide a tighter upper bound on deviation by using those already executed comparison views, together with the properties of the deviation function, as explained next.

We first outline the properties of our deviation function to provide a tighter bound on deviation. Specifically, according to Equation 1 any distance metric can be used for computing deviation, we take Euclidean distance as our metric. Let V_c and V_t be the comparison and target views with c categories, the squared Euclidean distance is:

$$dist^2_{P_{V_c},P_{V_t}} = \sum_{x=1}^{c} (P_{V_c}[x] - P_{V_t}[x])^2.$$

$$dist^2_{P_{V_c},P_{V_t}} = \sum_{x=1}^{c} P_{V_c}[x]^2 + \sum_{x=1}^{c} P_{V_t}[x]^2 - 2 \times \sum_{x=1}^{c} (P_{V_c}[x] \times P_{V_t}[x]) \quad (6)$$

Recall, in QuRVe, D_u is the theoretical maximum value of the distance between V_t and V_c. This maximum value is achieved when the last term in Equation 6 is zero and consequently:

$$dist^2_{P_{V_c},P_{V_t}} = \sum_{x=1}^{c} P_{V_c}[x]^2 + \sum_{x=1}^{c} P_{V_t}[x]^2$$

To be precise this maximum value is only possible when for each category x either $P_{V_c}[x]$ or $P_{V_t}[x]$ is zero. Accordingly,

$dist^2_{P_{V_c},P_{V_t}} = 2$ and the maximum deviation value $D_M = \sqrt{2}$. To maintain our multi-objective function (Eq. 4) normalized for all views, the deviation of each view is divided by that maximum deviation value D_M (Eq. 1), leading to the normalized upper bound on deviation to simply being $D_u = 1$.

However, this is the theoretical maximum, when the exact probability distribution of V_c and V_t are unknown. While, in our problem settings, P_{V_c} is known from the already executed comparison views for Q. Hence, we propose the uQuRVe scheme, which takes advantage of the already calculated P_{V_c} and calculates more realistic $D_u(V_{i,Q_j})$ and D_u.

Particularly, let the upper bound on deviation for a target view V_t corresponding to a comparison view V_c be $D_u[V_c]$. The main idea is to calculate the $D_u[V_c]$ for each comparison view and use it later for two purposes: 1) calculate upper bound on the utility of a target view, which can result in short circuiting of that view, and 2) calculate U_{Unseen}, which can result in early termination of the search.

Note that the query for the comparison view has been executed and its $P[V_c]$ is known. However, without executing the target query, we assume a hypothetical target probability distribution $P[V_t]$, such that it will result in the maximum value of deviation. Particularly, the upper bound $D_u[V_c]$ will be achieved when for the category in P_{V_c} having the minimum value, the corresponding value in P_{V_t} is maximum (i.e., 1.0). For this to happen, and since the overall $\sum_{x=1}^{c} P_{V_t}[x]$ has to be equal to 1.0, then all other values in P_{V_t} have to be 0.0. Consequently, all other categories in P_{V_t} are 0 and Eq. 6 is modified as:

$$dist^2_{P_{V_c},P_{V_t}} = \sum_{x=1}^{c} (P_{V_c}[x])^2 + \sum_{x=1}^{c} (P_{V_t}[x])^2 - 2 \times \min_{x \in c}(P_{V_c}[x]) \times 1 \quad (7)$$

For instance. assume a comparison view having four categories and $P_{V_c} = [0.3, 0.4, 0.1, 0.2]$. The minimum value is in the third category, hence, to have a maximum deviation let the hypothetical target view have $P[V_t] = [0, 0, 1, 0]$. Resultantly, the distance will be: $dist^2 = (0.3^2 + 0.4^2 + 0.1^2 + 0.2^2) + (1^2) - (2 \times 0.1 \times 1) = 1.1$, and the normalized upper bound on deviation $D_u[V_c]$ will be 0.7. Comparing that value to the theoretical upper bound used by QuRVe, which is 1, shows that in this example, our new upper bound is 30% less than the theoretical upper bound (i.e., significantly tighter bound).

Once $D_u[V_c]$ is calculated for all comparison views, $D_u = Max(D_u[V_c])$. D_u remains fixed for all iterations as the views are accessed in order of similarity value and there is no order on the deviation or upper bound of deviation of the views. Then U_{Unseen} is updated as $U_{Unseen} = \alpha_S \times S(Q, Q_j) + (1-\alpha_S) \times D_u$. Before executing the target query, the upper bound on utility of a view is calculated as $U(V_{i,Q_j}) \leq \alpha_S \times S(Q, Q_j) + (1 - \alpha_S) \times D_u(V_{i,Q_j})$. If $U(V_{i,Q_j}) \leq U_{Seen}$ then the target query is not executed and that view is pruned (short circuit). Clearly, uQuRVe is expected to performs better than QuRVe since the tighter upper bound on utility allows for more views to be short circuited and pruned, leading to a quicker early termination.

V_i	$S(V_i)$	$D(V_i)$	$D_u(V_i)$	$U(V_i)$	
V1	1	0	0.8	<= 0.9	0.55
V2	0.75	0.1	0.6	<= 0.675	0.425
V3	0.75	0.15	0.8	<= 0.775	0.45
V4	0.5	0.4	0.7	<= 0.6	0.45
V5	0.5		0.6	<= 0.55	
V6	0.5	0.7	0.8	<= 0.65	0.6
V7	0.25		0.6		
V8	0.25		0.7		

U_{Seen}	U_{Unseen}
0.55	0.775
0.55	0.775
0.55	0.65
0.55	0.65
0.6	0.525

FIGURE 6. The uQuRVe example.

Example: Consider the example of Figure 5b again. Now we have calculated the upper bound on deviation for all views as shown by column $D_U(V_i)$ in Figure 6, and $D_u = Max(D_u(V_i)) = 0.8$. Moreover, we added an additional column $U(V_i)$, in which the left side shows the upper bound on the utility of V_i, whereas the right side shows the actual utility of V_i if it is not pruned, and its actual deviation is to be computed. For instance, the figure shows that the utility of V_1 has to be less than an upper bound of 0.9 (i.e., $U(V_1) <= 0.9$), while its actual utility after it was processed was found to be 0.55. Similar to QuRVe, at the start $U_{Seen} = 0$ and $U_{Unseen} = 1$. The first view in S_{list} is V_1, initially it is checked for short circuit. Particularly, its upper bound of utility is computed as $U <= 0.5 + 0.5 \times 0.8 = 0.9$. As the upper bound is greater than U_{Seen}, therefore short circuit is not possible, and the view is probed for deviation and actual value of U is computed. After probing V_1 to V_4 in order of similarity, $U_{Seen} = 0.55$ and $U_{Unseen} = 0.65$. For V_5 when the upper bound on utility is computed using $D_u(V_5)$, it is equal to U_{Seen}. Therefore, V_5 is short circuited and its deviation is not probed. Afterwards, after probing V_6, U_{Unseen} becomes smaller than U_{Seen}, therefore the search is early terminated. In comparison to QuRVe which probed 8 iviews (as shown in Figure 5b), uQuRVe probed only 5 views.

In summary, both QuRVe and uQuRVe schemes maintain an S_{list}, which provides sorted(S) access, and D_{list} which provides random(R) access only. The D_{list} is accessed when the view under consideration can have a utility greater than U_{Seen}. Specifically, the access to D_{list} means calculating the statistical significance and eventually the deviation value for the view by generating the database probes for target query. Such probes constitute the cost incurred in finding and the recommending the top-k views. The QuRVe scheme reduces this cost by early terminating the search, based on the theoretical upper bound on U_{Unseen}. While uQuRVe further reduces this cost by: i) short circuiting the unnecessary low-utility views, and ii) allowing faster early termination by setting a tighter upper bound on U_{Unseen}. Clearly, that cost can be further minimized if the views with high utility are visited earlier in the search process, which results in setting U_{Seen} to a relatively higher value and achieve more pruning power. That is precisely the intuition underlying our pQuRVe scheme, which is presented next.

D. THE pQuRVe SCHEME

The QuRVe and uQuRVe schemes presented in the previous sections are based on the special case Sorted-Random (SR) model of the Threshold algorithm (TA). That is, only one objective is stored in a sorted list, whereas the second objective is accessed at random. In the context of our problem, that mapped to the similarity objective being calculated and sorted, whereas the deviation objective is accessed at random, and calculated on demand whenever a view V_i is not pruned, and its deviation value $D(V_i)$ is needed. Clearly, the SR model provided the benefit of eliminating the need to process pruned low-utility views, and in turn, save the query execution costs that would have been incurred to process them. However, as mentioned in Section IV-B, if the values of all the objectives in a top-k problem are readily available, then the SR model is outperformed by the more general model of the threshold algorithm (TA), in which all objectives are stored in sorted lists and each list also provides both sorted and random access [24], [25]. Hence, in this section we propose our new scheme, pQuRVe, with the goal of combining the benefits of both models.

The main idea underlying this scheme is to have more control on the ordering of probed views. Ideally, in a hypothetical oracle scheme, that ordering should lead to views with high utility to be probed first. While clearly that is not possible in a practical scheme, QuRVe and uQuRVe came close by ordering the views based on their similarity to the input query. Accordingly, in pQuRVe, instead of processing each view to compute its provided deviation, we rather utilize a reliable ''proxy'' for that deviation, as explained next.

Recall that the deviation $D(V_{i,Q_j})$, provided by a view V_{i,Q_j}, is measured as the deviation between the aggregate view V_i when generated from a selected subset of data D_{Q_j} vs. that view when generated from the entire database D (please see Eq. 3). Hence, intuitively, a refined query that selects a large subset of data is expected to generate views that are more similar to those generated from the entire database, which in turn exhibit low deviation. In other words, as a general trend, as the cardinality of the refined query result increases, the deviation of the views generated from that refined query generally decreases.

Given the intuition mentioned above, let Q_D be the query, which selects the whole dataset D. Further, assume Q_j is a refined input query. Hence, the more dissimilar (i.e., far) is Q_j from Q_D, the higher the expected deviation of the views generated from Q_j. Accordingly, for a view V_{i,Q_j} based on Q_j, we define a metric $O_d(V_{i,Q_j})$, which acts as a proxy for the deviation provided by V_{i,Q_j}. Particularly, $O_d(V_{i,Q_j})$ is defined in terms of the distance between V_{i,Q_j}'s underlying refined query (i.e., Q_j) and the query Q_D. To measure that distance, we use the distance metric defined in Equation 5 to specify $O(V_{i,Q_j})$ as:

$$O_d(V_{i,Q_j}) = s(Q_D, Q_j) \tag{8}$$

As mentioned in Section IV-B, computing $s(Q_D, Q_j)$ does not require any database probes, therefore it is computed for all views at negligible cost. Hence, in the pQuRVe scheme, an additional list named O_{list} is used to store the sorted O_d values for all views. As such, that O_{list} acts as a computationally inexpensive proxy for what would have been the sorted deviation list for those views.

As discussed earlier, differently from the SR model, in the optimized TA model, all the objective lists must provide both sorted and random access. Then TA proceeds iteratively in a round robin fashion, where in each iteration one of the lists is visited, and the top item in that list is identified using sorted access. For that item, all its objective scores are then fetched from all other lists using random access, and its overall objective score is then calculated.

Meanwhile, in our setting, the lists S_{list} and O_{list} provide precisely those sorted and random accesses. However, it is important to notice that a straight forward implementation of TA on those sorted lists will often lead to inaccurate results since O_{list} provides a proxy indicator of the deviation, but not the actual deviation itself. For instance, in the context of our problem, if a view is examined from the S_{list}, probing the O_{list} for its deviation will result in an incorrect overall utility score.

To overcome that limitation, we adapt the TA algorithm so that for a view V_{i,Q_j}, if it does not satisfy the pruning condition then: 1) if the unpruned V_{i,Q_j} is visited through the S_{list}, then its actual deviation is simply computed on demand using the D_{list} as in uQuRVe, and 2) if V_{i,Q_j} is visited from the O_{list}, then its similarity is probed from the S_{list} using random access, and if unpruned then its actual deviation is also computed on demand using the D_{list}. Hence, as in uQuRVe, the D_{list} is only accessed when required, whereas the two lists S_{list} and O_{list} are traversed according to the TA algorithm in a round robin fashion. That is, pQuRVe takes turns visiting the views across the two lists in descending order of their scores until a termination condition is reached. To detect early termination, similar to uQuRVe, U_{Unseen} and U_{Seen} are maintained and when $U_{Unseen} > U_{Seen}$ is false the search is terminated. However, under pQuRVe, a quick early termination is expected since the round robin traversal allows those views that are expected to have high-deviation to be visited early in the search process as they have a high chance of appearing at the top of the O_{list}.

Example: Consider again the same example shown in Figure 5b and further expanded here in Figure 7 to illustrate pQuRVe. Recall that the views $V_1 - V_8$ shown in figure are based on a set of refined queries $Q_1 - Q_8$. Further, in this example, assume that the analyst has no input query in mind, so she basically wants to analyze the entire database. Hence, in that case, the input query (i.e., Q_1) is simply equivalent to Q_D. Accordingly, as the figure shows, in the O_{list} the proxy deviation value for V_1 is 0. That is, for view V_1 which is based on the entire database, it is unexpected to find some interesting insights since the view will show the normal patterns exhibited by the entire database. As shown in figure, pQuRVe employs an S_{list} sorted on the similarity value and a O_{list} sorted on the proxy deviation O_d. Similar to our previous QuRVe schemes, at the start $U_{Seen} = 0$

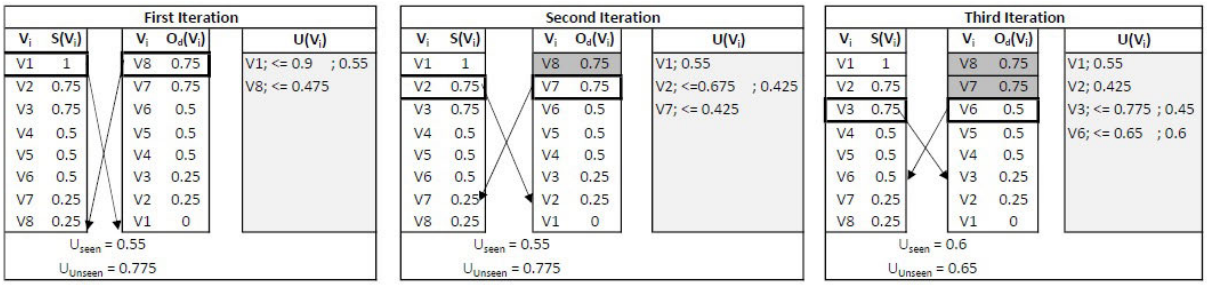


FIGURE 7. pQuRVe example.

and $U_{Unseen} = 1$. As discussed earlier, pQuRVe access the lists in round robin fashion similar to TA. Therefore, in the first iteration shown in Figure 7, the first view V_1 in S_{list} is visited. As V_1 does not satisfy the pruning and short circuit test, its utility is calculated by computing its actual deviation from d_{list}, as in uQuRVe. Consequently, U_{Seen} is updated by the value of $U(V_1) = 0.55$. Following the TA round robin traversal, pQuRVe alternates and visits the O_{list} list, in which V_8 is the top view. The upper bound on V_8's utility is computed, as shown in figure, $U(V_8) <= 0.475$. Hence, V_8 is pruned since its maximum possible utility (i.e., 0.475) is less than U_{Seen}. pQuRVe alternates back to S_{list}, where now V_2 is probed and its final utility value is computed as 0.425. Continuing with the round robin traversal, V_7 is accessed from the O_{list} list and is pruned. Then in the last round, pQuRVe probes V_3 from the S_{list} and V_6 from the O_{list}. After that point, no more views pass the upper bound check and eventually pQuRVe reaches early termination. As such, in this simple example, pQuRVe incurred the cost of processing only 4 views, as compared to uQuRVe requiring the processing of 5 views.

As shown in our experimental evaluation presented in the next section, pQuRVe typically achieves higher cost reductions than uQuRVe. This is because pQuRVe first visits the views which have high probability of having top-k utility values. Consequently, it sets U_{Seen} to a high value very quickly and the condition $U_{Unseen} > U_{Seen}$ becomes false very early as compared to uQuRVe. It is worth pointing out that the gains from pQuRVe become even more pronounced when the analyst is not particularly familiar with the analyzed data and make no initial hypothesis about expected insights (i.e., no input query is provided). In that case, the analyst would care most about finding interesting insights, and the similarity metric is irrelevant. Particularly, that setting is equivalent to $\alpha_S = 0$ and $\alpha_D = 1$. Consequently, the overall utility value completely depends on the deviation objective. In QuRVe, that maps to accessing all views in the unsorted D_{list}, therefore, it will actually perform same as the linear scheme. However, pQuRVe leverages the O_{list}, which partially ensures that views are accessed in pseudo order of high utility.

V. EXPERIMENTAL TESTBED

We perform extensive experimental evaluation to measure both the efficiency and effectiveness of our proposed QuRVe schemes. Here, we present the different parameters and settings used in our experiments.

TABLE 2. Summary of parameters.

Parameter	Range	Default
Similarity Weight (α_S)	0.0-1.0	0.5
Deviation Weight (α_D)	0.0-1.0 ($1-\alpha_S$)	0.5
top-k (k)	1-30	10
Grid resolution (γ)	$1/2^4$-$1/2$	$1/2^3$
Number of predicates (p)	1-9	3
Datasets	FLIGHTS, CENSUS	CENSUS
Number of input queries	10	10

Setup: We built a platform for refining query and recommending visualizations to evaluate the different search schemes 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.

Data Analysis: We assume a data exploration setting in which multi-dimensional datasets are analyzed. Table 2 lists down the parameters for data analysis, their default values, and ranges. We use *CENSUS:* the census income dataset [10] and *FLIGHTS:* the flight delays dataset [26]. The CENSUS dataset has 14 attributes and 48,842 tuples. The independent categorical attributes of the dataset are used as dimensions (e.g., occupation, work class, hours per week, sex, etc.), whereas the observation attributes are used as measures (capital gain, capital loss, etc.) and the numerical independent attributes are used for predicates (e.g., education, age, etc.). The same split of attributes is used for the FLIGHTS, which has 18 attributes and more than 5M tuples. For both datasets, in our default setting, $|\mathbb{A}| = 3$, $|\mathbb{M}| = 3$, $|\mathbb{F}| = 3$ and $p = 3$, where p is the number of predicates used in refinement, where the aggregate functions used are SUM, AVG and COUNT. The CENSUS dataset is used as the default dataset for our experiments.

In our analysis, α_S is set in the range $[0-1]$, where $\alpha_S + \alpha_D = 1$. In the default setting $\alpha_S = 0.5$, $k = 10$ and $\gamma = \frac{1}{2^3}$. For the purpose of statistical significance, in our experiments we use chi-square goodness of fit test, which is the standard test for comparing difference between sample and population data for categorical dimension attribute.

Schemes: While our work is the first to introduce query refinement for visual data exploration, we include in our

evaluation two baseline schemes that are closest to our work. As a baseline scheme, we include *SeeDB* [3], in which the goal is to maximize the overall utility of recommended views in terms of their deviations without attempting to refine the analyst's input queries. Additionally, we also include *Hill Climbing (HC)*, with halving search as another baseline method [27]. Aside from visual analytics, HC has been widely used for addressing the query refinement problem since it has been introduced in [27]. Our QuRVe schemes are compared to those baselines, as well as with the exhaustive *Linear* scheme presented in Section IV.

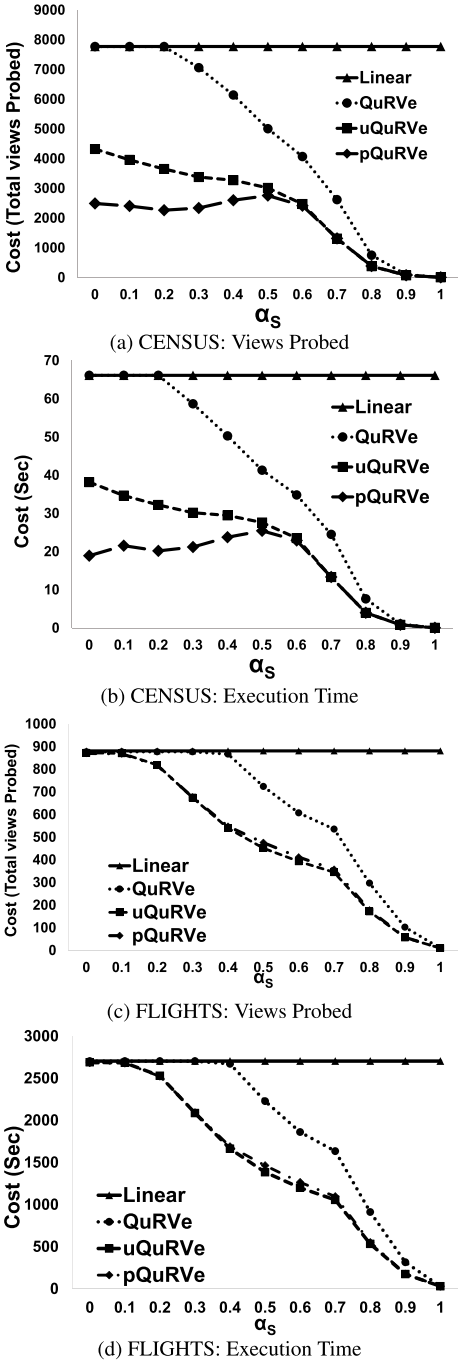
Performance: We evaluate the efficiency and effectiveness of the different recommendations schemes in terms of incurred query processing cost and their achieved overall utility. Particularly, as mentioned in Section III, the cost of a scheme is the total cost incurred in processing all the generated views. Hence, we use the total views probed and execution time as the cost metric. Meanwhile, also as mentioned in Section III, the achieved utility is the sum of the utilities of the recommended top-k views. It is worth mentioning that all our proposed QuRVe schemes provide the same utility, as they consider both the similarity and deviation objectives of the utility function. However, that is clearly not the case for the baseline schemes SeeDB and HC, where each considers only one of those objectives. In our evaluation, each experiment is performed with 10 randomly generated input queries, spread around the search space defined by the predicates in $\mathbb{P}$, then the average of the cost and overall utility is presented.

VI. EXPERIMENTAL EVALUATION

Impact of the α parameters (Figure 8): In this set of experiments, we measure the impact of the α values on cost (i.e., total number of views probed and execution time). Figure 8 shows how the cost of *Linear*, *QuRVe*, *uQuRVe* and *pQuRVe* schemes is affected by changing the values of α_S for the CENSUS and FLIGHTS datasets. In Figure 8, α_S is increasing while α_D is implicitly decreasing, for instance at $\alpha_S = 0$ the corresponding $\alpha_D = 1$ (i.e., $\alpha_D = 1 - \alpha_S$). Moreover, notice that all 4 of those schemes provide the same utility, hence, our focus in this experiment is on cost, whereas the results on utility are deferred to Figure 10.

For the Linear baseline scheme, Figure 8 shows that the Linear scheme has same the cost for all values of α_S, which is as expected since it performs exhaustive search over all combinations of refined queries, dimensions, measures and aggregate functions. Therefore, its cost depends on the number of all possible combinations, irrespective of the value of α_S.

As for the different measures of cost, Figure 8a shows cost in terms of number of views probed. It shows that *QuRVe* has almost the same cost as *Linear* for $\alpha_S = 0 - 0.2$, but outperform it as the value of α_S increases. This happens because in the *QuRVe* scheme, the upper bound on deviation is set to the theoretical maximum bound i.e., $D_u = 1$ and when $\alpha_S = 0$, $U_{Unseen} = 0 \times S + 1 \times D_u = 1$, consequently early termination is not possible. On the contrary, as α_S increases,



(a) CENSUS: Views Probed

(b) CENSUS: Execution Time

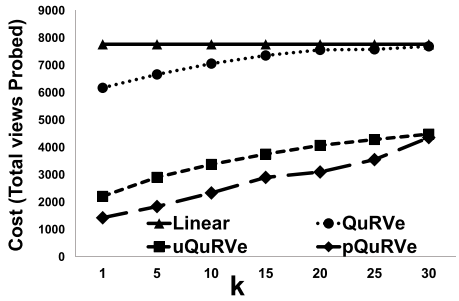
(c) FLIGHTS: Views Probed

(d) FLIGHTS: Execution Time

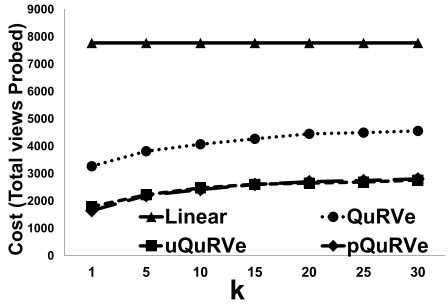
FIGURE 8. Impact of α_S and α_D on cost.

chances of applying the early termination condition based on the similarity value becomes possible. Consequently, this prunes many of the database probes.

The amount of achieved pruning is further increased for $\alpha_S \leq 0.8$ under *uQuRVe*, because of its tighter upper bound on deviation. Consequently, this results in earlier early termination and increased number of short circuits on deviation calculation. For instance, in Figure 8 at $\alpha_S = 0.3$, *uQuRVe* shows around 50% reduction in cost as compared to *QuRVe*. The cost is further reduced by *pQuRVe*, which is able to

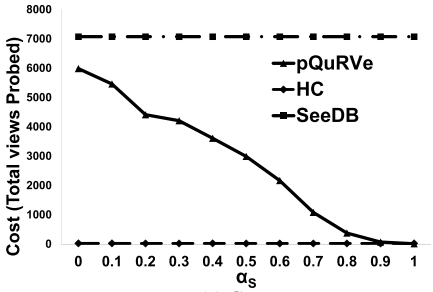


(a) $\alpha_S = 0.3, \alpha_D = 0.7$

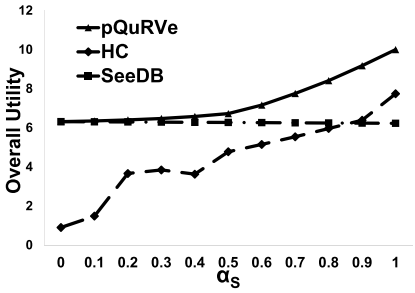


(b) $\alpha_S = 0.6, \alpha_D = 0.4$

FIGURE 9. Impact of k on cost at $\alpha_S = 0.3$, $\alpha_D = 0.7$ and $\alpha_S = 0.6$, $\alpha_D = 0.4$.



(a) Cost



(b) Overall Utility

FIGURE 10. Impact on effectiveness.

prune deviation calculations even more because it visits the most promising top-k views first before low utility views. For instance, Figure 8 shows that *pQuRVe* reduces the processing cost by more than 35%, compared to *uQuRVe*, at $\alpha_S = 0.2$. Notice *uQuRVe* and *pQuRVe* are able to prune views for all values of α due to the tighter upper bound on utility.

Figure 8b shows the cost for the CENSUS dataset in terms of execution time. The figure clearly shows that the execution time pattern exhibited by all schemes is the same as Figure 8a. Naturally, this is because the execution time is I/O-bound and it is directly proportional to the number of probed views.

Figures 8c and 8d show the cost in terms of execution time and number of views probed for the FLIGHTS dataset. Figure 8d shows that the execution time for *Linear* is rather high as compared to Figure 8b because of the larger size of the dataset. Moreover, the figure also shows that *uQuRVe* has lower execution time compared to the *QuRVe* scheme for all values of α_S due to the optimized upper bounds. However, for $\alpha_S \leq 0.5$ *QuRVe* performs similar to linear because the dataset lacks high deviation views. Particularly, across all the possible views in that dataset, the highest exhibited deviation is 0.25, which results in low value of U_{Seen} and consequently less early termination opportunities. Note that the cost of *uQuRVe* and *pQuRVe* is almost the same for the FLIGHTS dataset, differently from the CENSUS dataset. This is because for the FLIGHTS dataset, the low-cardinality high-deviation views turned out to be statistically insignificant, and as a result *pQuRVe* converged to perform similar to *uQuRVe* in terms of pruning power.

Impact of k (Figure 9): In the previous experiments, the value of k is set to 10 (i.e., top-10 views are recommended). Figures 9b and 9b show the sensitivity of our schemes to different values of k, while keeping α_S and α_D fixed. Particularly, Figures 9a and 9b show that the *Linear* scheme is insensitive to the increase in the value of k. This is because it visits all views and sort them according to their utility irrespective of the value of k.

For Figure 9a, $\alpha_S = 0.3$, *QuRVe* performs better than *Linear* for small values of k. However, as k increase it almost performs same as *Linear*, because *QuRVe* uses loose upper bounds on deviation and with higher values of k the early termination does not get a chance to trigger. Figure 9a also shows that *uQuRVe* has lower cost than *QuRVe* for all values of k. This is because as soon as *uQuRVe* has seen the top-k highest utility views, early termination will be enabled leading to pruning many unnecessary low utility views. For instance, in case of top-1, *uQuRVe* reduces cost by up to 65% compared to the *Linear* scheme. Moreover, *pQuRVe* performs even better than *uQuRVe* for all values of k as it utilizes the proxy to deviation to increase its pruning power. In Figure 9b, $\alpha_S = 0.6$, all *QuRVe* schemes perform multiple folds better than *Linear*, and the performance remains stable even with the increasing number of k. This is because, for higher values of α_S, the similarity objective starts to dominate the overall utility function, while the deviation objective receives less weight, leading to favorite conditions for pruning and early termination.

Impact on Effectiveness ((Figure 10): As mentioned in Section V, we also evaluate the performance of the baselines SeeDB and Hill climbing (HC) in comparison to our QuRVe schemes. Particularly, in this experiment we first focus on the effectiveness provided by those baselines in terms of the achieved overall utility. Since all the QuRVe schemes provide the same effectiveness, we only include the results from *pQuRVe*, which is also the most efficient QuRVe scheme in terms of minimizing processing costs.

Figures 10a and 10b show the cost and overall utility of all the mentioned schemes. Figure 10a shows that *SeeDB* has the same high cost for all values of α_S, as it is only designed to maximize deviation, and is incognizant to similarity. Therefore, its cost depends on the number of all possible combinations, irrespective of the values of α_S. Figure 10a also shows that the *HC* scheme provides the same cost regardless of α_S value. This is because *HC* is a local search based scheme, which is expected to hit a local maxima, hence, always has low cost. Meanwhile, the cost of *pQuRVe* decreases with the increase in α_S due to the early terminations and short circuits of the scheme.

Figure 10b shows SeeDB provides a constant utility, which is significant lower than *pQuRVe* for $\alpha \geq 0.5$. This is because SeeDB is purely based on the deviation metric, and is oblivious to similarity. Hence, when α_D starts decreasing, the views selected by SeeDB would lack similarity to the input query, and the overall utility starts to decrease as compared to the overall utility provided by *pQuRVe*. Meanwhile, our hybrid objective utility function is incorporated in *HC*, which leads to improving the utility as α_S increases. However, as mentioned earlier, *HC* is a local search greedy scheme, therefore, it typically provides a local maxima solution and falls short in achieving the global maxima. To the contrary, the *pQuRVe* scheme achieves the maximum utility value for all values of α_S.

Scalability - Impact of Dimensionality (Figure 11): We note that the search space for our problem depends on $|A|$, $|M|$, $|F|$, the number of predicates p, and grid resolution γ,. Increasing any of those factors naturally leads to increasing the search space. Consequently, the processing cost incurred by all schemes increases as there are more views that are visited in search for the top-k views.

Figure 11 shows the impact of the number of dimensions on cost. Particularly, for this experiment the number of dimensions ($|A|$) is increased from 1 to 9, while in the previous experiments the number of dimensions was set to 3. As expected, Figure 11 shows that the increase in the cost of *Linear* is linear with the increase in $|A|$. However, for the *QuRVe* family, the increase in cost occurs at a much slower rate. It is worth noting that in addition to the number of dimensions, the data characteristics of each dimension also have an impact on the cost *QuRVe*. Particularly, in this experiment we observed that the cost of *QuRVe* and *uQuRVe* increases as the number of dimensions is increased from 1 to 3, because the schemes search from more views to generate top-k views. However, as we added more dimension and reached $|A| = 5$, the cost of both schemes started to decrease. This drop in cost happened because the newly introduced dimensions generated views with high deviation, which in turn resulted in increasing the value of U_{Seen} and triggering early termination.

We also note a special case, in which the cost of *pQuRVe* scheme is higher than that of *uQuRVe* for $|A| = 1 - 3$. The reason for that discrepancy is that *pQuRVe* attempts to find high-deviation views early in the search. However, when the dataset lacks such high-deviation views, as it has been the case with $|A| = 1 - 3$, then *pQuRVe* is expected to experience some overhead. Particularly, *pQuRVe* as it traverses the S_{list} and O_{list} in round robin, it will be probing views from the O_{list} in attempt to quickly find those high-deviation views. In the absence of such views with significantly high deviation, *pQuRVe* might end up probing more views in comparison to *uQuRVe*. However, in the typical case where the dataset contains some interesting high deviation views, *pQuRVe* consistently outperforms *uQuRVe*, as it is shown in the figure where $|A| \geq 5$.

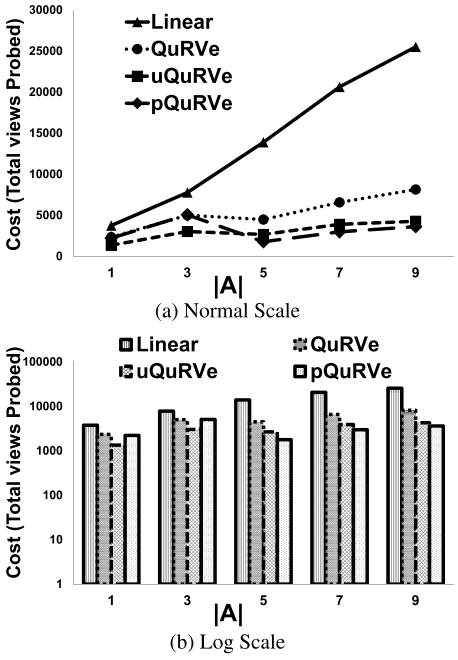


FIGURE 11. Impact of dimensions on cost.

Scalability - Impact of Grid Resolution (Figures 12–13): As mentioned in Section IV, to enable the refinement of continuous dimensions, those dimensions are discretized by a user specified parameter γ. Figures 12 and 13 show the impact of this discretization factor, or grid resolution, on the cost and overall utility value achieved by all schemes. Note that a large value of γ represents a sparse grid with wider cells, which means there are fewer refined queries to explore. Alternatively, as γ decreases, the grid cell width decreases, which means the possible number of refined queries increases. More refined queries generally implies increase in cost but it should also be able to improve the overall utility of the top-k views. Figure 12 clearly shows that as the grid becomes dense the cost of the *Linear* scheme increase as it has to search through a larger number of refined queries for top-k views. At the same time, the reduction in cost by the *QuRVe* schemes are more prominent for dense grids (i.e., low γ). This is because the number of possible refined queries increases and it gives the *QuRVe* schemes more opportunities to prune unnecessary views. Figure 12b shows the same results of Figure 12a by using a log scale on the y-axis to further underscore the performance gains

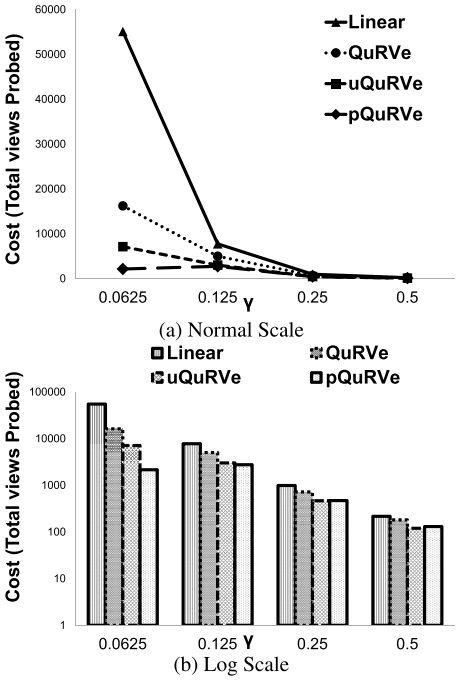


FIGURE 12. Impact of grid resolution on cost (log scale).

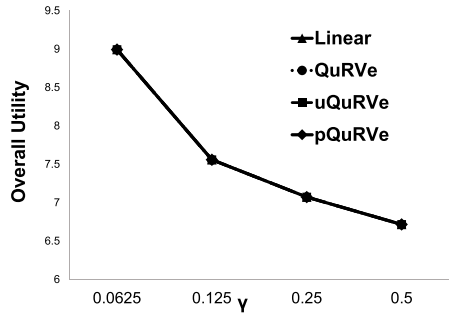


FIGURE 13. Impact of grid resolution on overall utility.

provided by *QuRVe*. For instance, the figure clearly shows that at $\gamma = 0.0625$, the cost of *pQuRVe* is more than 10 folds less than that for *Linear*. Finally, Figure 13 shows the achieved overall utility (i.e., the sum of the utility of top-k views). Clearly, all four schemes provide the same overall utility as all schemes recommend the same top-k views. However, it can be clearly seen that as γ decreases the overall utility increases, since a denser grid allows for a high resolution query refinement, and in turn discovering those particular subsets of data that are both: 1) most similar to the analyzed data subset, and 2) reveal interesting insights in terms of providing high deviation.

VII. RELATED WORK

In this section, we present our review of related work in the areas of visualization recommendation, query refinement, and hypothesis testing.

Visualization Recommendation: Many efforts in the visual data exploration domain have been directed towards the development of effective tools and techniques for visualization recommendation. On the one hand, some recent research efforts have focused on learning human perception for recommending interesting visualizations. In particular, the focus is understanding which visualization and transformation is good under which scenarios and using that learning to mine for interesting visualizations [28]–[30]. However, such systems involve heavy interaction with the user in the learning phase, hence these systems are less effective when users are rapidly exploring unfamiliar or complex datasets. On the other hand, data-driven efficient techniques have been developed for recommending top-k visualizations based on some utility metric such as deviation, similarity, correlation, outliers etc. [3], [4], [6], [7], [9], [12]. The main idea underlying those solutions is to automatically generate all possible views of the explored data, and recommend the top-k *interesting* views, where the interestingness of a view is quantified according to a pre-specified objective function that captures the utility of the view. Our work proposed in this paper advances the data-driven visualization recommendation one step further towards full automation by not only recommending the top-k interesting views, but also recommending the queries that lead to such views. Specifically, in this work, we explore all possible subsets of data for discovering interesting views. Orthogonal to recommending visual aggregations, which is the focus of our work, exploring a data search space to find those interesting data subsets has been the focus of some recent research efforts. For instance, the work in [31] models the behavior of data subsets by progressively adding filters. The objective of that work is to recommend a network of visualizations within a subset of data where interestingness is quantified in terms of the distance between the parent and the child visualizations. Meanwhile, the work in [32] investigates another version of the problem, in which LOF (local outlier factor) is the employed utility metric. Accordingly, it explores all possible subsets of data specified by a single predicate only, where such predicate is defined on a categorical attribute.

Query Refinement: In this work, we propose to automatically refine an input exploratory query for the objective of view recommendation. In the domain of data exploration, automatic query refinement has been used on different flavors of problems that involve generating refined queries with answers that satisfy some cardinality constraints. For instance, it has been used in works focused on cardinality, similarity and aggregate constraints [13], [14], [17], [27], [33], exploring predicates to answer why questions and to explain outliers [15], [16], to solve empty query result problem [34], [35]. The cardinality constrained based query refinement is a computationally expensive problem, therefore heuristics that efficiently return approximate results were developed. However, in our case those heuristics, such as Hill Climbing, fall short in meeting our goal to rank views optimally not approximately. Moreover, differently from the simple objective of finding queries that satisfy a certain constraint on cardinality result, our goal is to search for

those queries that generate views, which maximize our hybrid multi-objective utility function. Further for the cardinality constrained based query refinement problem, an interactive model of refinement was proposed that incorporated user feedback to capture user preferences for the refined queries [13]. That work addressed a relaxed problem compared to ours, as only partial combinations of refinements were generated due to interactivity with the user. In comparison, the core of our work is around the automated data-driven objective, however we also combine the user preference and other control parameters of the proposed schemes. Scorpion, similar to us, is a system to analyze large datasets through aggregates [15]. However, while our QuRVe refines predicates to find interesting aggregate views, Scorpion takes a set of outlier points in an aggregate query result as input and finds predicates that explain those outliers [15]. The properties of aggregate operators are used to derive the search and prune the predicates search space. SAQR [17] proposed a scheme for similarity-aware refinement of aggregate queries, which satisfy the aggregate and similarity constraints imposed on the refined query and maximize its overall utility. Another query refinement work that considers the similarity to the input query has been introduced in [14]. However, the goal of that work is to generate a set of alternative queries to meet the constraint on similarity to the original query and the aggregate constraint. Meanwhile, our goal is to generate alternative queries that maximize our multi-objective utility function, which is based on similarity to the original query, as well as the deviation from the comparison dataset.

Hypothesis testing: Hypothesis testing is a well studied area in the domain of statistics. Moreover, it also recently found several applications in the domain of data exploration in terms of determining the statistical significance of discovered insights. For instance, the work in [36] is one of the initial efforts towards exploratory hypothesis testing, which enabled researchers to use computational methods to examine large numbers of hypotheses and to identify those that have a reasonable chance of being true. The proposed framework generates subsets using existing frequent pattern mining techniques, pair the subsets to form tentative hypotheses, and rank the statistically significant hypotheses in ascending order of their p-value. Additionally, [36] introduces restricted monte carlo tests for validating correlations of datasets, where initial experiments indicate that those significance tests can improve the data discovery process. Finally, the work in [19] automatically creates hypothesis based on aggregate queries and performs significance testing based on chi-square tests. However, [19] focuses on controlling the rate of false discoveries, while our proposed schemes, in addition to eliminating false discoveries, they also allow efficient navigation of the views search space.

VIII. CONCLUSION

Motivated by the need for visualizations recommendation that leads to interesting discoveries and avoid common pitfalls such as random or false discoveries, in this paper we formulated the problem of query refinement for view recommendation and proposed the QuRVe schemes for view recommendation. QuRVe refines the original query in search for more interesting views. It efficiently navigates the refined queries search space to maximize utility and reduce the overall query processing cost. That cost is further reduced with the uQuRVe scheme, which applies tight upper bounds to prune more views. Further, we also proposed the pQuRVE scheme, which navigates the refined queries search space in an order that resembles a utility-based sorted order of the views. Our experimental results show that employing the QuRVe schemes offer significant reduction in terms of the costs incurred in recommending top-k aggregate views.

REFERENCES

[1] E. Wu, L. Battle, and S. R. Madden, "The case for data visualization management systems," *Proc. VLDB Endowment*, vol. 7, no. 10, pp. 903–906, Jun. 2014.

[2] S. Idreos, O. Papaemmanouil, and S. Chaudhuri, "Overview of data exploration techniques," in *Proc. SIGMOD*, 2015, pp. 277–281.

[3] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis, "See DB: Efficient data-driven visualization recommendations to support visual analytics," *Proc. VLDB Endowment*, vol. 8, no. 13, pp. 2182–2193, Sep. 2015.

[4] R. Ding, S. Han, Y. Xu, H. Zhang, and D. Zhang, "Quickinsights: Quick and automatic discovery of insights from multi-dimensional data," in *Proc. SIGMOD*, 2019, pp. 317–322.

[5] Ç. Demiralp, P. J. Haas, S. Parthasarathy, and T. Pedapati, "Foresight: Recommending visual insights," *Proc. VLDB Endowment*, vol. 10, no. 12, pp. 1937–1940, Aug. 2017.

[6] T. Sellam and M. Kersten, "Ziggy: Characterizing query results for data explorers," *Proc. VLDB Endowment*, vol. 9, no. 13, pp. 1473–1476, Sep. 2016.

[7] H. Ehsan, A. Mohamed Sharaf, and K. Panos Chrysanthis, "Muve: Efficient multi-objective view recommendation for visual data exploration," in *Proc. ICDE*, 2016, pp. 731–742.

[8] H. Ehsan, "View recommendation for visual data exploration," Ph.D. dissertation, School Inf. Technol. Elect. Eng., Univ. Queensland, Brisbane, QLD, Australia, 2019.

[9] C. Wang and K. Chakrabarti, "Efficient attribute recommendation with probabilistic guarantee," in *KDD*, 2018, pp. 2387–2396.

[10] *Adult Data Set*. Accessed: Jul. 2019. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/adult

[11] H. Ehsan, A. Mohamed Sharaf, and G. Demartini, "Qurve: Query refinement for view recommendation in visual data exploration," in *Proc. ADBIS*, 2020, pp. 154–165.

[12] H. Ehsan, M. A. Sharaf, and P. K. Chrysanthis, "Efficient recommendation of aggregate data visualizations," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 2, pp. 263–277, Feb. 2018.

[13] C. Mishra and N. Koudas, "Interactive query refinement," in *EDBT*, 2009, pp. 862–873.

[14] M. Vartak, V. Raghavan, A. Elke Rundensteiner, and S. Madden, "Refinement driven processing of aggregation constrained queries," in *Proc. EDBT*, 2016, pp. 101–112.

[15] E. Wu and S. Madden, "Scorpion: Explaining away outliers in aggregate queries," *Proc. VLDB Endowment*, vol. 6, no. 8, pp. 553–564, Jun. 2013.

[16] Q. T. Tran, "How to conquer why-not questions," in *Proc. SIGMOD*, 2010, pp. 15–26.

[17] A. Albarrak, A. Mohamed Sharaf, and X. Zhou, "SAQR: An efficient scheme for similarity-aware query refinement," in *Proc. DASFAA*, 2014, pp. 120–125.

[18] D. Stephen Bay and J. Michael Pazzani, "Detecting group differences: Mining contrast sets," *Data Min. Knowl. Discov.*, vol. 5, no. 3, pp. 213–246, 2001.

[19] Z. Zhao, L. D. Stefani, E. Zgraggen, C. Binnig, E. Upfal, and T. Kraska, "Controlling false discoveries during interactive data exploration," in *Proc. SIGMOD*, 2017, pp. 527–540.
[20] Y. Chung, "Towards quantifying uncertainty in data analysis & exploration," *IEEE Data Eng. Bull.*, vol. 41, no. 3, pp. 15–27, Oct. 2018.
[21] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. New York, NY, USA: Academic, 1977.
[22] A. Telang, "One size does not fit all: Toward user and query dependent ranking for Web databases," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 9, pp. 1671–1685, Oct. 2012.
[23] A. Kadlag, A. V. Wanjari, J. Freire, and J. R. Haritsa, "Supporting exploratory queries in databases," in *DASFAA*, 2004, pp. 594–605.
[24] A. Marian, "Evaluating top-K queries over Web-accessible databases," *ACM Trans. Database Syst.*, vol. 29, no. 2, pp. 319–362, 2004.
[25] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *J. Comput. Syst. Sci.*, vol. 66, no. 4, pp. 614–656, Jun. 2003.
[26] *2015 Flight Delays and Cancellations*. Accessed: Jul. 2019. [Online]. Available: https://www.kaggle.com/usdot/flight-delays
[27] N. Bruno, S. Chaudhuri, and D. Thomas, "Generating queries with cardinality constraints for DBMS testing," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 12, pp. 1721–1725, Dec. 2006.
[28] Y. Luo, X. Qin, N. Tang, and G. Li, "Deepeye: Towards automatic data visualization," in *Proc. ICDE*, 2018, pp. 101–112.
[29] V. Dibia and C. Demiralp, "Data2 Vis: Automatic generation of data visualizations using Sequence-to-Sequence recurrent neural networks," *IEEE Comput. Graph. Appl.*, vol. 39, no. 5, pp. 33–46, Sep. 2019.
[30] K. Z. Hu, A. Michiel Bakker, S. Li, T. Kraska, and A. César Hidalgo, "VizML: A machine learning approach to visualization recommendation," in *Proc. CHI*, 2019, pp. 1–12.
[31] D. J. L. Lee, H. Dev, H. Hu, H. Elmeleegy, and G. A. Parameswaran, "Avoiding drill-down fallacies with *VisPilot* assisted exploration of data subsets," in *Proc. IUI*, 2019, pp. 186–196.
[32] T. Matsumoto, Y. Sasaki, and M. Onizuka, "Data slice search for local outlier view detection: A case study in fashion EC," in *Proc. Workshops EDBT/ICDT*, 2019, pp. 1–5.
[33] A. M. Albarrak and M. A. Sharaf, "Efficient schemes for similarity-aware refinement of aggregation queries," *World Wide Web*, vol. 20, no. 6, pp. 1237–1267, Nov. 2017.
[34] I. Muslea and J. T. Lee, "Online query relaxation via Bayesian causal structures discovery," in *Proc. AAAI*, 2005, pp. 831–836.
[35] N. Koudas, C. Li, K. H. A. Tung, and R. Vernica, "Relaxing join and selection queries," in *Proc. VLDB*, 2006, pp. 199–210.
[36] F. Chirigati, H. Doraiswamy, T. Damoulas, and J. Freire, "Data polygamy: The many-many relationships among urban spatio-temporal data sets," in *Proc. SIGMOD*, 2016, pp. 1011–1025.