Received 11 April 2023, accepted 7 May 2023, date of publication 6 June 2023, date of current version 23 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3283457

RESEARCH ARTICLE

Efficient Diversification for Recommending Aggregate Data Visualizations

MOHAMED A. SHARAF[1], RISCHAN MAFRUR[2], AND GUIDO ZUCCON[2]
[1]Department of Computer Science and Software Engineering, United Arab Emirates University, Al Ain, United Arab Emirates
[2]School of Information Technology and Electrical Engineering (ITEE), The University of Queensland, Saint Lucia, QLD 4067, Australia

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

The work of Mohamed A. Sharaf was supported by the United Arab Emirates (UAE) University Strategic Research Program under Grant 12R147. The work of Rischan Mafrur was supported by the Indonesia Endowment Fund for Education [Lembaga Pengelola Dana Pendidikan (LPDP)] under Grant 201706220111044. The work of Guido Zuccon was supported in part by the Australian Research Council Discovery Early Career Researcher Award (DECRA) Research Fellowship under Grant DE180101579 and in part by the Google Faculty Award.

ABSTRACT Visual data exploration is ubiquitous in nearly every industry and organization to support discovering data-driven actionable insights. However, unlocking those insights requires analysts to manually construct a prohibitively large number of aggregate queries and visually explore their results, looking for those valuable and insightful visualizations. Such a challenge naturally motivated the development of novel solutions that automate the visual exploration process, and recommend to analysts those particular queries that best visualize their data and reveal interesting actionable insights. In such automated solutions, there is a clear need for providing analysts with a diversified and concise set of recommended visualizations, which cover and represent a large combinatorial high-dimensional space of possible visualizations. However, directly incorporating existing diversification methods leads to a ''process-first-diversify-next'' approach, in which all possible data visualizations are generated first through executing a large number of aggregate queries. To address this challenge and minimize the incurred query processing costs, in this work, we propose novel optimization techniques for the efficient diversification of recommended insightful visualizations. The key idea underlying our proposed techniques is to identify and eliminate the processing of a large number of low-utility insignificant visualizations. Meanwhile, for the potentially high-utility insightful visualizations, shared multi-query optimization techniques are proposed for further reduction in data processing cost. Our extensive experimental evaluation on real datasets demonstrates the performance gains provided by our proposed techniques, in terms of minimizing the query processing cost (i.e., efficiency), as well as maximizing the quality of recommendations (i.e., effectiveness).

INDEX TERMS Data exploration, visual analytics, recommendation diversification.

I. INTRODUCTION

Visual data exploration is an essential step in the data science pipeline, in which analysts examine datasets up-close to extract valuable insights (e.g., [1], [2], [3], [4]). This process has been traditionally performed manually, where the analyst interactively applies various exploratory queries (such as SQL-based filtering, aggregation, joins, etc.). The results of those queries are presented as data-driven visualizations (e.g., bar or line charts, scatter plots, etc.). The analyst then examines those visualizations looking for insights, which are used as a springboard to decide their next analytical query. In that process, analysts need to manually construct a prohibitively large number of queries and visually explore their results looking for insights, which is clearly an ad-hoc and labor-intensive process.

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

The challenges mentioned above motivated multiple research efforts that focused on automatic recommendation for data exploration. That is, recommender systems dedicated to provide the user with suggestions for specific, high-utility exploratory queries [5], [6], [7], [8], [9], [10], [11], [12], [13], [14]. Such systems are data-driven (also known as discovery-driven) systems, which use heuristic notions of *interestingness* and employ them in the recommendation. The main idea underlying those solutions is to automatically generate all possible exploratory queries of the data, generate their corresponding visualizations, and recommend the top-k interesting ones. Meanwhile, the interestingness of a query is quantified using some *utility* metric over its result.

A large body of 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 (e.g., [6], [7], [8], [12], [13], [14], [15]). Basically, such data-driven deviation-based systems recommend visualizations based on reference data or reference visualizations. The underlying premise is that a visualization is likely to be interesting if it displays a large deviation from some reference (e.g., complete dataset, another dataset, or the rest of the database). Particularly, the deviation-based metric measures the distance between each aggregate view generated from the reference dataset (i.e. reference view) and the view generated from the analyzed data subset (i.e. target view). That is, the deviation-based metric measures the pairwise distance between all the possible aggregations over the selected data and recommends the ones with highest deviation between target and reference view. The intuition is that a view with high deviation (i.e., importance score) is expected to reveal some important insights that are specific to the data subset under analysis.

However, one drawback of that utility-based approach is that it often recommends similar views, leaving the data analyst with a limited amount of gained insights. To address that limitation, in our previous work [11], we proposed the DiVE schemes to eliminate redundancy and provide full coverage of the possible insights to be discovered by the recommendation process. Towards this, the DiVE schemes employ a hybrid objective utility function, which captures both the importance together with the diversity of the insights revealed by the recommended visualizations.

However, our work in [11] shows that directly applying diversification methods to insight recommendation leads to a "process-first-diversify-next" approach [16], in which all possible data visualizations are generated first through executing a large number of aggregate queries. To address this challenge and minimize the incurred query processing cost, in [11], we have proposed pruning-based techniques for minimizing the number of processed aggregate queries. Particularly, the main underlying idea is to leverage the properties of both the importance and diversity to eliminate the processing of a large number of low-utility views.

In this work, we expand on our original DiVE scheme along two orthogonal dimensions, namely, 1) providing high-quality recommendations, and 2) supporting scalable shared processing of aggregate views. Specifically, our original DiVE scheme partially relies on an adaptive pruning technique that samples the space of possible views to estimate the maximum possible utility score to be achieved by any view. Naturally, that sampling method introduces some approximation to the recommendation process, and in turn, the quality of recommendation becomes dependent on the sampling size. Hence, in this work, we expand on that method, by introducing a rectifying algorithm that is integrated in our DiVE scheme and allows to automatically adapt the sample size as necessary to improve the quality of recommendation. Our rectifying algorithm employs back-tracking and caching mechanisms that allow for minimizing the query processing costs, while maximizing the accuracy of recommendations (Section IV-B). Additionally, to further reduce query processing time, we propose a sharing-based optimization, which processes different overlapping aggregate views simultaneously at a minimal cost. Specifically, instead of each query underlying a view being processed independently, such optimization leverages the similarity between those queries, and utilize multi-query processing, in which the execution of overlapping queries is shared (Section V-A).

Further, we note that both query pruning and multi-query optimization aim to reduce the query processing cost incurred during the recommendation process, but they achieve this goal through different approaches. Particularly, query/view pruning relies on eliminating the processing of some queries, while multi-query optimization is based on the simultaneous processing of many overlapping queries. Consequently, we propose a novel hybrid approach that combines both adaptive pruning with sharing-based optimization. Specifically, our approach utilizes adaptive pruning to identify high-utility views, which are then grouped and processed simultaneously using sharing-based optimization. Our hybrid approach provides two benefits: 1) eliminating the processing of some queries when early termination is reached through pruning, and 2) reducing the processing time of unpruned queries by utilizing shared processing (Section V-B). Finally, we conduct an extensive experimental evaluation on real datasets, comparing the performance and illustrating the benefits achieved by our proposed algorithms (Section VII).

The main contributions of this work are summarized as follows:

- We formulate the problem of recommending views that are both important and diverse based on a multi-objective utility function (**Section II**).
- We introduce our DiVE scheme, which employs novel pruning techniques that leverage the salient characteristics of our objective function to minimize the query processing cost incurred in view recommendation while maximizing the quality of recommendation (**Section III**).
- We propose an approximate version of DiVE, which employs adaptive sampling to further reduce the solution search space, together with a rectifying mechanism that automatically tunes the sampling size to maximize the quality of recommendation (**Section IV**).

- We extend DiVE to incorporate a sharing-based optimization, which allows processesing different overlapping aggregate views simultaneously at a minimal cost (**Section V**).
- We conduct an extensive experimental evaluation on real datasets, which compares the performance of various algorithms and demonstrates the benefits achieved by our proposed methods (**Section VII**).

II. PRELIMINARIES

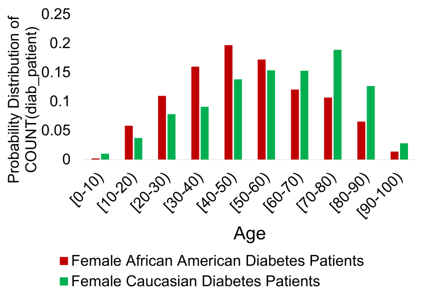
In this section, we first present a motivating example, which demonstrates the need for diversification in the process of visualization recommendation (Sec. II-A). Then, we present our model for recommending diversified visualizations (Sec. II-B), together with a formulation of the problem addressed in this work (Sec. II-C).

A. MOTIVATING EXAMPLE

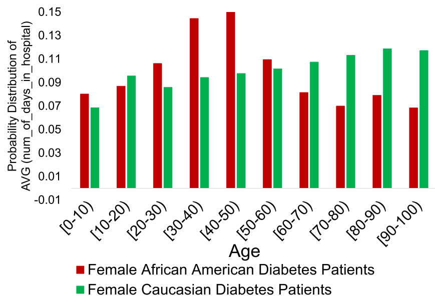
Consider a data analyst trying to gain some insights into the Diabetes 130-US hospitals dataset [17]. The dataset represents 10 years (1999-2008) of clinical care at 130 US hospitals and includes 100,000 diabetes patients. The data contains multiple attributes such as gender, age, race, time in the hospital, number of outpatients, inpatients, etc.

Since many studies have been conducted to understand the interplay between the diabetes disease and some demographic factors, such as gender, race, etc. (e.g., [18], [19]), our analyst might also be interested in exploring the Diabetes dataset to find some interesting data-driven insights along that direction. For instance, the analyst might choose to conduct a comparison between Female African American diabetes patients versus Female Caucasian diabetes patients. To accomplish this, the analyst writes an SQL query that selects female diabetes patients who are African American (i.e., `race=African American AND gender=Female`) as the target data subset for analysis, and the female Caucasian diabetes patients as the reference data subset (i.e., `race = Caucasian AND gender = Female`). Since the analyzed data contains different dimensions (e.g., *age*, *race*, *gender*, *readmitted*) and different measures (e.g., *time_in_hospital*, *number_inpatient*, *number_outpatient*), it can be challenging for the analyst to manually select the combinations of dimensions and measures that reveal interesting insights. Hence, to automatically recommend interesting bar chart visualizations, a recommendation system (e.g., [6], [7], [8], [12], [13], [14], [15]) would apply different SQL aggregate functions to the views generated from all possible pairwise combinations of dimensions and measures, then the most important views are presented to the analyst.

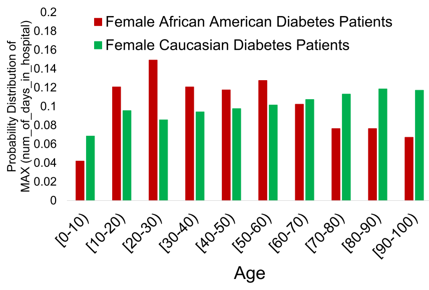
Figure 1 shows the top-1, top-2, and top-3 recommended views according to the deviation-based metric (e.g., [6], [7], [8], [12], [13], [14], [15]). Interestingly, all three visualizations show a clear discrepancy between the two patients groups across the different age brackets. For instance, consider the top-1 recommendation shown in Figure 1a,



(a) Top1: `COUNT`(diabetes patients) vs. race in different age



(b) Top2: `AVG`(number of days in hospital) diabetes patients vs. race in different age



(c) Top3: `MAX`(number of days in hospital) diabetes patients vs. race in different age

FIGURE 1. Top three recommended visualizations.

which plots the normalized distribution of the number of diabetic patients in each patient group across the different age brackets. As the visualization shows, for Caucasian females, the majority of diabetic patients are above the age of 50, with a high concentration of patients in the [70-80] age range. However, for African American patients, the distribution of patients by age looks significantly different. Particularly, Figure 1a shows that the distribution of patients over age follows more of a normal distribution, in which the highest concentration of patients is in the rather younger [38], [39], [40], [41], [42], [43], [44], [45], [46], [47], [48] age range. That discrepancy is further underscored in the top-2 and top-3 recommended visualizations shown in Figure 1b and 1c, which are based on the *num_of_days_in_hospital* measure.

However, comparing all three recommended visualizations, it is easy to notice their similarity. Particularly, all three convey more or less the same insight, they are based on the same dimensional attribute (i.e., *patient age*), and two of which are also based on the same measure attribute (i.e., *number of days in hospital between admission and discharge*). Despite that obvious similarity between those different visualizations, they are still recommended together to the analyst due to the high deviation between the target and reference views exhibited by each visualization.

B. RECOMMENDING DIVERSIFIED VISUALIZATIONS

Similar to existing work (e.g., [3], [6], [7], [8], [20]), we assume a visual data exploration session, which starts with an analyst submitting a query Q on a multi-dimensional database D_B. Essentially, Q selects a subset D_Q from D_B by specifying a query predicate T. Hence, Q is defined as: Q: `SELECT * FROM` D_B `WHERE` T`;`. For example, Q: `SELECT * FROM` *diabetes_db* `WHERE race= African American;`

Ideally, the analyst would like to generate some aggregate views (e.g., bar charts or scatter plots) that unearth some valuable insights from the selected subset D_Q. However, achieving that goal is only possible if the analyst knows exactly what to look for!

Hence, the goal of existing works, such as [3], [5], [6], [7], [8], [20], [21], [22], and [23], is to automatically recommend such aggregate views. To specify and recommend such views, we consider a multi-dimensional database D_B, which consists of a set of dimensional attributes $\mathbb{A}$ and a set of measure attributes $\mathbb{M}$. Also, let $\mathbb{F}$ be a set of possible aggregate functions over the measure attributes. Hence, specifying different combinations of dimension and measure attributes with various aggregate functions, generates a set of possible views $\mathbb{V}$ over the selected dataset D_Q. Particularly, an aggregate view V_i is specified by a tuple $< A_i, M_i, F_i >$, where $A_i \in \mathbb{A}$, $M_i \in \mathbb{M}$, and $F_i \in \mathbb{F}$.

Clearly, manually looking for insights in each view $V_i \in \mathbb{V}$ is a labor-intensive and time-consuming process. Such challenge motivated multiple research efforts that focused on automatic recommendation of views based on some metrics that capture the utility of a recommended view (e.g., [5], [6], [7], [8], [9], [10], [21], [22], [23], [24], [25]). Towards that, data-driven metrics are employed to capture the interestingness or importance of a recommended view. Recent case studies have shown that a deviation-based metric is effective in providing analysts with important views that highlight some of the particular trends of the analyzed datasets (e.g., [6], [7], [8], [11], [15], [22]).

1) CONTENT-DRIVEN IMPORTANCE

Essentially, the deviation-based metric compares an aggregate view generated from the selected subset dataset D_Q (i.e., target view $V_i(D_Q)$) to the same view if generated from a reference dataset D_R (i.e., reference view $V_i(D_R)$). That is, it measures the deviation between the result of $V_i(D_Q)$ and that of $V_i(D_R)$. Consequently, from a visualization point of view, that deviation is a content-based metric that captures the

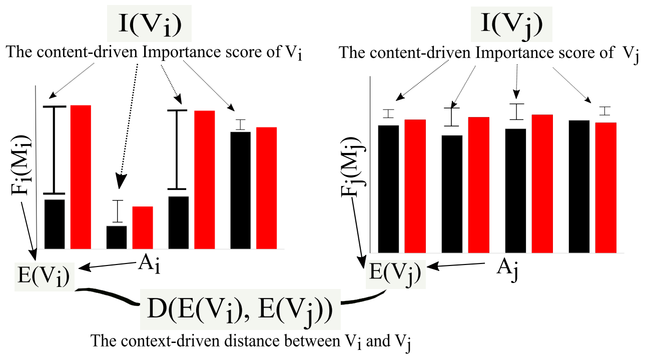

I(Vi)
The content-driven Importance score of Vi
I(Vj)
The content-driven Importance score of Vj
Fi(Mi)
Fj(Mj)
E(Vi)
Ai
E(Vj)
Aj
D(E(Vi), E(Vj))
The context-driven distance between Vi and Vj

FIGURE 2. Content vs. Context of views.

difference between the content of the visualization generated by $V_i(D_Q)$ vs. the visual content of $V_i(D_R)$. The premise underlying the deviation-based metric is that a view V_i that results in a high deviation is expected to reveal some important insights that are very particular to the subset D_Q and distinguish it from the patterns in D_R. For instance, the visualization of number of diabtes patients vs. race in different age extracted from D_Q are fundamentally different from number of diabtes patients vs. race in different age extracted from D_R.

To calculate the content-based deviation, each target view $V_i(D_Q)$ is normalized into a probability distribution $P[V_i(D_Q)]$ and similarly, each reference view into $P[V_i(D_R)]$. Then, the importance score of V_i is measured in terms of the distance between $P[V_i(D_Q)]$ and $P[V_i(D_R)]$ (as illustrated in Figure 2), and is simply defined as:

$$I\left(V_i\right) = dist\left(P\left[V_i(D_Q)\right], P\left[V_i(D_R)\right]\right) \tag{1}$$

where $I\left(V_i\right)$ is the importance score of V_i and *dist* is a distance function. Similar to existing work [6], [7], [8], [20], we adopt a Euclidian distance, but other distance measures are also applicable (Earth Mover's distance, K-L divergence, etc.).

2) CONTEXT-DRIVEN SIMILARITY

While recommending views based on their importance has been shown to reveal some interesting insight, it also suffers from the drawback of recommending similar and redundant views, which leaves the data analyst with a limited scope of possible insights. As illustrated, Figure 1 shows recommended visualizations that basically reveal the same insight.

To address that limitation, in this work we posit that employing diversification techniques (e.g., [16], [26], [27], [28], [29], [30], [31], [32]) in the process of view recommendation allows eliminating that redundancy and provides a good and concise coverage of the possible insights to be discovered.

Diversity has been well known and widely used in recommendation systems for maximizing information gain and minimizing redundancy (e.g., [29], [30], [31], [33]). At a high level, diversity essentially measures how different (i.e., diverse) are the individual data objects within a set. It is important to notice that central to that computation is some notion of distance measure between data objects.

Meanwhile, our previous work [11] was the first to consider diversity in the context of aggregate data visualizations. Particularly, in [11] we propose a metric to quantify the (dis)similarity between the distinct features of different visualizations. As discussed in [11], our metric is based on considering the query underlying each view (i.e., the query executed to create the view). Hence, in addition to the data-driven content-based deviation metric, we also adopt a query-driven context-based deviation metric as shown in Figure 2.

Particularly, in addition to the content of a view V_i which is described by its probability distribution (i.e., $P(V_i)$) as defined in Sec. II), we also consider the context of the view $E(V_i)$, which is defined in terms of the query underlying V_i as: $E(V_i) = \{A_i, M_i, F_i\}$.

Such definition of view context leads to a special case of the existing work on query recommendation [34], [35], [36], in which the normalized distance between two queries is simply measured using the Jaccard similarity measure. Hence, the Jaccard similarity between two aggregate views V_i and V_j is measured as: $Sim\left(V_i, V_j\right) = \frac{\mid E(V_i) \cap E(V_j) \mid}{\mid E(V_i) \cup E(V_j) \mid}$.

We note that the jaccard similarity assigns equal weights to each of the element in a set. Accordingly, when applied to aggregate views, then two views with the same attribute and different measure and aggregate function will have the same similarity score as any other pair of views with same measure but different attribute and aggregate function. However, an analyst may consider two views with the same attribute A_i more similar than two views with same measure attribute M_i. To allow the analyst to specify such preference, each contextual component of a view is associated with a weight that specifies its impact on determining the (dis)similarity between views. Specifically, for any view V_i, let $w(e)$ be the weight assigned to the e^{th} context component of $E(V_i)$. Since, $E(V_i)$ is a set of three components $\{A_i, M_i, F_i\}$, then $\sum_{e=1}^{3} w(e) = 1$. Accordingly, the similarity between any two views V_i and V_j is measured as:

$$J(V_i, V_j) = \frac{\sum_{e \in E(V_i) \cap E(V_j)} w(e)}{\sum_{e \in E(V_i) \cup E(V_j)} w(e)}$$

Consequently, the context-based deviation between V_i and V_j is calculated as:

$$D\left(V_i, V_j\right) = 1 - Sim\left(V_i, V_j\right) \tag{2}$$

3) HYBRID OBJECTIVE FUNCTION

Given the set of all possible views $\mathbb{V}$, our goal is to recommend set $S \subseteq \mathbb{V}$, where $|S| = k$. Hence, to capture the quality of recommendation, our hybrid objective function is designed to consider both the importance and diversity of the recommended views. As such, the importance score of S is calculated as the average value of the importance measure of each view in S, as given in Eq. 1. In particular, according to Eq. 1 any distance metric can be used for computing deviation, in this work, we utilize Euclidean distance as our distance metric. Hence, the total importance score of S is defined as: $I(S) = \sum_{i=1}^{k} \frac{I(V_i)}{I_u}$, $V_i \in S$, where I_u is the upper bound on the importance score for an individual view.

To compute that upper bound I_u, let V_i and V_j be our target and reference views respectively with c categories and h is for each category, then the squared Euclidean distance is: $dist^2_{PV_i,PV_j} = \sum_{h=1}^{c} (PV_i[h] - PV_j[h])^2$, then $dist^2_{PV_i,PV_j} = \sum_{h=1}^{c} PV_i[h]^2 + \sum_{h=1}^{c} PV_j[h]^2 - 2 \times \sum_{h=1}^{c} (PV_i[h] - PV_j[h])$. The maximum value of the Euclidean distance is achieved when the last term is zero and it results $dist^2_{PV_i,PV_j} = \sum_{h=1}^{c} PV_i[h]^2 + \sum_{h=1}^{c} PV_i[h]^2$. In particular, the maximum value I_u is only possible when for each category h either $PV_i[h]$ or $PV_j[h]$ is zero. As the result, $dist^2_{PV_i,PV_j} = 2$ and $dist_{PV_i,PV_j} = \sqrt{2}$. Thus, $I_u = \sqrt{2}$ is used to normalize the average importance score for set S.

In order to achieve diversity when selecting a set of objects, several diversity functions have been employed in the literature, with special focus on the MaxMin and MaxSum functions (e.g., [16], [27], [28], [29], [30], [31], [32]). Particularly, the goal in MaxSum is to maximize the average pairwise distance between the objects in a set, whereas in MaxMin the goal is to maximize the minimum pairwise distance between those objects. In this work, we consider both notions of diversity when selecting a set of recommended views.

Accordingly, given a distance metric $D\left(V_i, V_j\right)$, as given in Eq. 2, the MaxSum diversity of a set S can be simply measured as follows:

$$f(S, D) = \frac{1}{k(k-1)} \sum_{i=1}^{k} \sum_{j>i}^{k} D\left(V_i, V_j\right), V_i, V_j \in S.$$

Since the maximum context-based deviation between any two views in Eq. 2 is 1.0, dividing the sum of distances by $k(k-1)$ ensures that the diversity score of set S is normalized and bounded by 1.0.

Different from MaxSum, recall that the objective of MaxMin is to maximize the minimum context-based deviation of a set S. Hence, the MaxMin diversity of a set S can be simply measured as follows: $f(S, D) = min\ D\left(V_i, V_j\right), V_i, V_j \in S$, where S is the selected subset of views and $D\left(V_i, V_j\right)$ is the distance metric as given in Eq. 2.

C. PROBLEM DEFINITION

Putting it together, for a set of views $S \subseteq \mathbb{V}$, our hybrid objective function is formulated as the linear weighted combination of the importance score, $I(S)$ and diversity score $f(S, D)$, and is defined as:

$$O(S) = (1 - \lambda) \times I(S) + \lambda \times f(S, D) \tag{3}$$

where $0 \leq \lambda \leq 1$ is employed to control the preference given to the importance and diversity components.

Hence, our goal is to find an optimal set of views S, which maximizes the objective function $O(S)$, and is formally defined as follows:

Definition 1 (Recommending Diversified Important Views): Given a target subset D_Q and a reference subset D_R, the goal is to recommend a set $S \subseteq \mathbb{V}$, where $|S| = k$, and $\mathbb{V}$ is the set of all possible target views, such that the overall hybrid objective $O(S)$ is maximized.

1) COST OF VISUALIZATION RECOMMENDATION

Exisiting research has shown that recommending aggregate data visualizations based on data-driven content-based deviation is a computationally expensive task [6], [7], [8], [11], [20]. Moreover, integrating diversification into the view recommendation problem, as described above, further increases that computational cost. In particular, the incurred processing cost includes the following two components: 1) Query processing cost C_Q: measured in terms of the time needed to execute and compare all the queries underlying the set of target views as well as their corresponding reference views (i.e., content-based deviation), and 2) View diversification cost C_D: measured in terms of the time needed to compute all the pairwise distances between each pair of target views (i.e., context-based deviation). Consequently, the total cost C_T for recommending a set of views is simply defined as: $C_T = C_Q + C_D$.

In principle, traditional data diversification methods that consider both relevance and diversity can be directly applied in the context of our problem to maximize the objective function defined in Eq. 3. However, those setting assume that the relevance of an object is either given or requires simple computation. To the contrary, in our setting for view recommendation, the importance of a view is a computationally expensive operation, which requires the execution of a target and reference view. To address that challenge and minimize the incurred query processing cost, next we propose our DiVE scheme, which leverages the properties of both the importance and diversity to prune a large number of low-utility views, without compromising the quality of recommendations.

III. THE DiVE SCHEMES

In our previous work [11], we have proposed the DiVE scheme for maximizing our hybrid objective utility function outlined above, while minimizing the incurred query processing cost. DiVE incorporates a set of different algorithms, namely: *DiVE-Greedy, DiVE-iSwap, and DiVE-dSwap*. Among those algorithms, DiVE-dSwap has been shown to provide the best performance in terms of both effectiveness and efficiency [11]. Hence, in this paper, we propose further extensions for improving the performance of our DiVE-dSwap method. For the sake of completeness, and to facilitate explaining our new extensions, next we briefly summarize the DiVE-dSwap method, as it was proposed in [11].

Algorithm 1 DiVE-dSwap

```
    Input: Set of views $\mathbb{V}$ and result set size $k$
    Output: Result set $S \subseteq \mathbb{V}$, $|S| = k$
 1  $S \leftarrow$ set of maximum diversity
 2  $X \leftarrow [\mathbb{V} \backslash S]$
 3  $O_{current} \leftarrow 0$
 4  $improve \leftarrow True$
 5  while $improve = True$ do
 6      for $X_i$ in set $X$ do
 7          $S' \leftarrow S$
 8          for $S_j$ in set $S$ do
 9              if $O(S') < O(S \backslash S_j \cup X_i)$ then
10                  $S' \leftarrow S \backslash S_j \cup X_i$
11              end
12          end
13          if $O(S') > O(S)$ then
14              $S \leftarrow S'$
15          end
16      end
17      if $O(S) > O_{current}$ then
18          $O_{current} \leftarrow O(S)$
19          $improve \leftarrow True$
20      else
21          $improve \leftarrow False$
22      end
23  end
24  return S
```

A. THE DiVE-dSwap SCHEME

Our DiVE-dSwap scheme falls under the local search type of algorithms. In particular, a local search algorithm starts out with a complete initial solution and then attempts to find a better solution in the neighborhood of that initial one. Like constructive algorithms, local search algorithms are also widely used in solving optimization problems including diversification. For instance, the Swap local search method has been utilized to maximize diversity [28], [30], [32]. The basic idea underlying DiVE-dSwap is to start with an initial set S of size k and then iteratively modify the set S in order to improve the value of the objective function $O(S)$. In our DiVE-dSwap, S is initialized with the k views that maximize diversity. Then, DiVE-dSwap iteratively interchanged a view from X to S (Algorithm 1 line 14) until no more views can be swapped between X and S (Figure 3). To make that selection, DiVE-dSwap assigns a score to each view in X, which is based on the hybrid objective function $O(S)$, as defined in Eq. 3. Specifically, the utility score assigned to a view $X_i \in X$ is computed as:

$$U(X_i) = (1 - \lambda) \times I(X_i) + \lambda \times setDist(X_i, S \backslash S_j) \quad (4)$$

$$setDist(X_i, S \backslash S_j) = \frac{1}{|S \backslash S_j \cup X_i|} \sum_{\substack{j=1 \\ V_j \in S \backslash S_j \cup X_i}}^{|S \backslash S_j \cup X_i|} D(V_i, V_j)$$

Thus, in each iteration, the view with highest utility score is selected and interchanged to S, until no more views can be

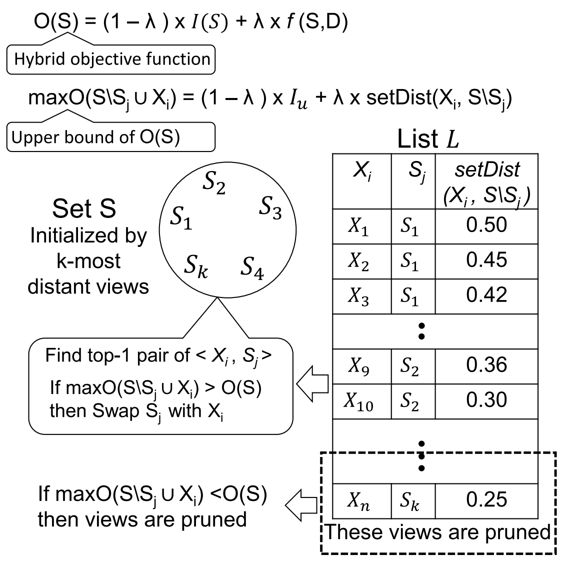

O(S) = (1 – λ) x I(S) + λ x f (S,D)
Hybrid objective function
maxO(S\S_j ∪ X_i) = (1 – λ) x I_u + λ x setDist(X_i, S\S_j)
Upper bound of O(S)
Set S
Initialized by k-most distant views
S_1 S_2 S_3 S_k S_4
List L
X_i
S_j
setDist (X_i, S\S_j)
X_1 S_1 0.50
X_2 S_1 0.45
X_3 S_1 0.42
X_9 S_2 0.36
X_10 S_2 0.30
X_n S_k 0.25
Find top-1 pair of < X_i, S_j >
If maxO(S\S_j ∪ X_i) > O(S) then Swap S_j with X_i
If maxO(S\S_j ∪ X_i) <O(S) then views are pruned
These views are pruned

FIGURE 3. Pruning condition in DiVE-dSwap.

swapped between X and S, which is reached when no further improvement is achieved in the value of O (Algorithm 1 line 17).

We note that the overall cost of DiVE-Swap is $C_T = C_Q + C_D$, where C_Q is the query processing cost (i.e., data-driven), and C_D is the cost for computing Jaccard distances (i.e., query-driven), as described in Sec. II. Clearly, C_Q is equal to the number of possible views and is $O(n)$, where n is the number of possible views, whereas C_D can reach up to $O\left(kn^2\right)$, where k is the number of recommended views.

B. PRUNING FOR DiVE-dSwap

As described above, DiVE-dSwap execute all the underlying queries for each view $X_i \in X$. However, only a small fraction of those views is actually included in the final top-k recommended set. Consequently, a significant amount of query processing cost is incurred for generating low-utility views. Thus, in this section, we propose efficient techniques for pruning such low-utility views without incurring the high cost for evaluating their importance score.

Recall that under DiVE-dSwap, in each iteration a view $X_i \in X$ is selected to replace a view $S_j \in S$. The criterion for that selected view is to improve $O(S)$. That is, $O(S \backslash S_j \cup X_i) > O(S)$. Hence, the task is to find that top-1 pair of views $< X_i, S_j >$ that provides the maximum improvement in $O(S)$ once interchanged. Without pruning, that requires iterating through S and X simultaneously and computing O for each pair, which requires processing and generating each view in X. To avoid such expensive processing, we propose a view pruning technique, which is described next, and illustrated in Figure 3.

To enable view pruning, a list L is created for all possible swap pairs $< X_i, S_j >$, where L is sorted based on the diversity achieved if the swap is to be made (as shown in

Algorithm 2 Pruning Algorithm

```
Input: Set of views $\mathbb{V}$ and result set size $k$
Output: Result set $S \subseteq \mathbb{V}$, $|S| = k$
$S \leftarrow$ set of maximum diversity
$X \leftarrow [\mathbb{V} \backslash S]$
$PI \leftarrow 0.80$
$O(S) \leftarrow$ objective function of $S$ (Eq. 3)
$\bar{I}_{au} \leftarrow AdaptivePruning(PI)$
Function setDist($[X_i, S \backslash S_j]$):
    $diversity \leftarrow D(X_i, S \backslash S_j)$;
    return $diversity$
End Function
$L \leftarrow [X_i, S \backslash S_j, setDist\,[X_i, S \backslash S_j]]$
$L \leftarrow$ sorted by $setDist$ DESC
$maxO(X_i, S \backslash S_j) \leftarrow (1 - \lambda) \times \bar{I}_{au} + \lambda \times setDist\,[X_i, S \backslash S_j]$
Function AdaptivePruning($PI$):
    execute some views according to $PI$
    if the number executed views satisfies $PI$ then
        $\bar{I}_{au} \leftarrow$ get the highest score of $I$
    else
        execute more views until $PI$ is satisfied
        $\bar{I}_{au} \leftarrow$ get the highest score of $I$
    end
    return $\bar{I}_{au}$
End Function
while $improve = True$ do
    for $X_i$ in set $X$ do
        $S' \leftarrow S$
        for $S_j$ in set $S$ do
            if $O(S') < O(S \backslash S_j \cup X_i)$ then
                $S' \leftarrow S \backslash S_j \cup X_i$
            end
            if $maxO(X_i, S \backslash S_j) < O(S)$ then
                $prune\,[X_i, S \backslash S_j]$
            end
            if more views are executed then
                $I_{au} \leftarrow$ get the highest score of $I$
                if $I_{au} > \bar{I}_{au}$ then
                    use $I_{au}$ instead of $\bar{I}_u$ to calculate
                      $maxO$
                    if $maxO(X_i, S \backslash S_j) < O(S)$ then
                        $prune\,[X_i, S \backslash S_j]$
                    end
                end
            end
        end
        if $O(S') > O(S)$ then
            $S \leftarrow S'$
        end
    end
    if $O(S) > O_{current}$ then
        $O_{current} \leftarrow O(S)$
        $improve \leftarrow True$
    else
        $improve \leftarrow False$
    end
end
return S
```

Figure 3). Particularly, L is sorted by *setDist* $(X_i, S \setminus S_j)$, where *setDist* $(X_i, S \setminus S_j) = \frac{1}{|S \setminus S_j \cup X_i|} \sum_{j=1, V_j \in S \setminus S_j \cup X_i}^{|S \setminus S_j \cup X_i|} D(V_i, V_j)$ (Algorithm 2 line 6-11). It is important to notice that up to this point the only processing needed is to compute diversity without any query execution to evaluate the importance of any X_i. Given that setting, the task of finding the best swap is clearly similar to top-k query processing, for which numerous optimization techniques are proposed (e.g., [37], [38]). Particularly, to find the top-1 swap, each view X_i is initially assigned an importance equal to the upper bound I_u (Sec. II). In turn, the upper bound on $O(S)$ achieved by incorporating X_i is computed as: $maxO(S \setminus S_j \cup X_i)$, which is based on the actual diversity achieved by the swap, and the upper bound on importance. As such, $maxO(S \setminus S_j \cup X_i)$ is compared against $O(S)$, leading to one of the following two cases: If $maxO(S \setminus S_j \cup X_i) > O(S)$, then the swap $< X_i, S_j >$ can "potentially" improve $O(S)$. Hence, at that stage the view X_i needs to be generated in order to evaluate its actual importance $I(X_i)$. Otherwise, the pair $< X_i, S_j >$ is pruned if: $maxO(S \setminus S_j \cup X_i) < O(S)$ (Algorithm 2 line 30-39).

As shown in Figure 3, the set S is initialized by k-most distant views and in each iteration, the S_j is swapped with X_i if $maxO(S \setminus S_j \cup X_i) > O(S)$. Simply put, if the upper bound $maxO$ achieved by that swap is still less than the current $O(S)$, then the actual $O(S \setminus S_j \cup X_i)$ is guaranteed to be less than $O(S)$ and the pair $< X_i, S_j >$ can be safely ignored and pruned. More importantly, since the L is sorted by diversity, and if $maxO(S \setminus S_j \cup X_i) < O(S)$, then the next views are also guaranteed to provide no improvement and that iteration of DiVE-dSwap reaches early termination. Hence, for all the remaining views no query processing is needed, which significantly reduces the overall cost. In Figure 3, the views that have been pruned after early termination are shown within a dotted rectangle.

IV. PREDICTIVE INTERVAL FOR ADAPTIVE BOUNDS

In general, the pruning schemes provided by DiVE-dSwap rely on the fundamental idea of evaluating the upper bound of the benefit provided by a view V_i towards the objective O. If that maximum benefit is still not enough to consider V_i to join S, then V_i is pruned and its query processing cost is saved. Moreover, to evaluate that upper bound, DiVE-dSwap computes the actual diversity offered by V_i and instead of computing its actual importance $I(V_i)$, it is substituted with the maximum attainable importance score I_u.

Naturally, overestimating $I(V_i)$ leads to overestimating the importance of V_i and consequently limited pruning power is achieved. Meanwhile, for most datasets, I_u as it is computed in Sec. II is in fact an overestimation of the importance achieved by any view V_i. Hence, our goal in this section is to provide a tighter bound on $I(V_i)$, which allows for maximum pruning while maintaining the quality of the solution. To address this challenge, we propose our method DiVE-dSwap-Adaptive (Sec. IV-A), which is able to provide a tighter upper bound on I_u by processing some sample views. To further reduce the number of processed sample views while providing that tighter bound, we additionally propose DiVE-dSwap-Rectify(PI), which expands on the basic DiVE-dSwap-Rectify by employing a novel rectifying mechanism (Sec. IV-B).

A. ADAPTIVE PRUNING

In this section, we propose our adaptive pruning method, DiVE-dSwap-Adaptive, which relies on estimating a tight upper bound on the importance score of each view V_i. Estimating such bound will in turn lead to more pruning power than that achieved when the importance score of each view is estimated as the maximum theoretical bound I_u (as described in the previous section).

Recall, I_u the upper bound on deviation is the theoretical maximum value of the distance between $P[V_i(D_R)]$ or $P[V_i(D_Q)]$. This maximum value is achieved when for each category either in $P[V_i(D_R)]$ or $P[V_i(D_Q)]$ is zero. Hence, I_u is a theoretical bound for the maximum importance achieved by any view in any dataset. For most real datasets, however, that condition is rarely satisfied and the actual upper bound, denoted as I_{au}, is typically much smaller than I_u.

Meanwhile, a "hypothetical" pruning scheme that utilizes that actual upper bound I_{au} is expected to deliver more pruning power than the schemes using the theoretical upper bound I_u, especially when $I_{au} \ll I_u$. In practice, however, that hypothetical scheme is not achievable since obtaining the value I_{au} requires executing all the possible views, which is clearly in conflict with the goal of pruning.

Accordingly, rather than using overestimated I_u or obtaining the actual I_{au}, our goal is to estimate I_{au} with high accuracy and minimum number of query executions. In particular, given the set of possible views $\mathbb{V}$, the goal is to estimate the maximum importance $\bar{I}_{au}$ given by some view in $\mathbb{V}$. However, estimating the maximum value of a population is known to be a challenging problem, as opposed to estimating other statistics such as average or sum [39].

Thus, instead of estimating I_{au}, we rely on non-parametric predictive interval models to determine its value with certain level of confidence without any assumption on the population [39]. To apply that predictive model, some sample views are executed and the maximum importance observed in that sample is recorded as $\bar{I}_{au}$. To determine the number of samples, a Predictive Interval (PI) is to be defined, such that: $PI = \frac{(m-1)}{(m+1)}$, where m is the number of samples.

For instance, setting $PI = 90\%$ (i.e., PI 0.90), results in $m = 19$. That is, after processing a set of 19 sample views and setting $\bar{I}_{au}$ to the maximum importance observed in that set, then $\bar{I}_{au}$ is actually higher than the importance value of any unsampled view V_i with probability 90%. Or the other way around, there is only a 10% chance that the importance value of an unseen view V_i will be higher than the maximum importance seen so far (i.e., $\bar{I}_{au}$), which renders $\bar{I}_{au}$ a tight upper bound on $I(V_i)$ with 90% confidence. Clearly, the higher the value of PI, the higher the accuracy

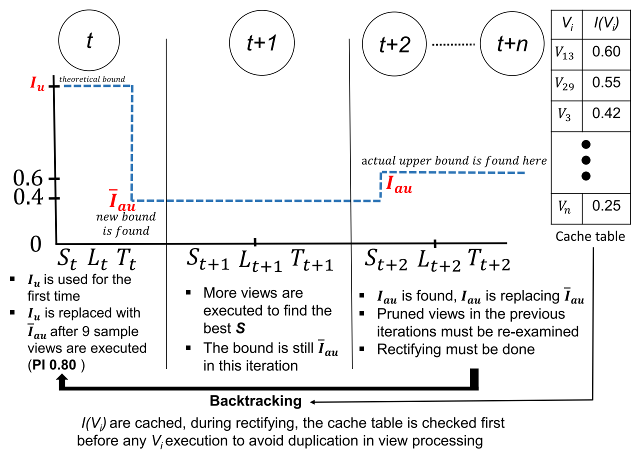

t
t+1
t+2
t+n
I_u
theoretical bound
0.6
0.4
0
$\bar{I}_{au}$
new bound is found
actual upper bound is found here
$\bar{I}_{au}$
V_i | $I(V_i)$
V_{13} 0.60
V_{29} 0.55
V_3 0.42
V_n 0.25
Cache table
S_t L_t T_t
I_u is used for the first time
I_u is replaced with $\bar{I}_{au}$ after 9 sample views are executed (PI 0.80)
S_{t+1} L_{t+1} T_{t+1}
More views are executed to find the best S
The bound is still $\bar{I}_{au}$ in this iteration
S_{t+2} L_{t+2} T_{t+2}
I_{au} is found, I_{au} is replacing $\bar{I}_{au}$
Pruned views in the previous iterations must be re-examined
Rectifying must be done
Backtracking
$I(V_i)$ are cached, during rectifying, the cache table is checked first before any V_i execution to avoid duplication in view processing

FIGURE 4. The updating of upper bound in each iteration of DiVE-dSwap-Adaptive(0.80).

of $\bar{I}_{au}$, which also requires sampling (i.e., processing) more views.

Figure 4 shows an example of augmenting our *DiVE-dSwap* scheme with our proposed adaptive method with $PI = 80$ (i.e., *DiVE-dSwap-Adaptive(0.80)*. As the figure shows, in the first iteration of *DiVE-dSwap*, the upper bound on the importance of any view V_i is set to the theoretical maximum attained importance I_u. However, going through the normal steps of *DiVE-dSwap*, some views are processed to compute their actual importance. Once enough views are processed to meet the number of samples required for $PI 0.80$, then $\bar{I}_{au}$ is used (Algorithm 2 line 13). Particularly, for this examples, $PI 0.80$ is achieved after 9 views are processed (i.e., $m = 9$). Furthermore, in this example, $\bar{I}_{au}$ is set to 0.4, which is the maximum importance observed when processing those 9 views. That new upper bound of 0.4 will allow early termination and pruning of some low-utility views in the first iteration, as well as subsequent iterations (Algorithm 2 line 30-39).

B. ADAPTIVE PRUNING WITH RECTIFYING

As expected, our experimental results presented in Sec. VII show that the pruning power provided by our adaptive pruning method (i.e., *DiVE-dSwap-Adaptive(PI)*) described above is inversely correlated to PI. That is, high pruning is achieved at low values of PI. But also as expected, at low PI, the effectiveness of recommendation (i.e., $O(S)$) is often reduced. That is because low PI requires only a small number of processed sample views, which leads to low accuracy in estimating the upper bound on importance $\bar{I}_{au}$. This might motivate using higher values for PI, such as PI 0.95 or PI 0.97, to achieve a higher accuracy in upper bound estimation. However, it also requires executing more views which is clearly in conflict with our goal of minimizing the number of query executions. Hence, the challenge addressed in this section is maintaining a high quality in recommendation ((i.e., $O(S)$), while executing a small number of views (i.e., low PI).

To illustrate that challenge, reconsider again our example from the previous section, which is illustrated in Figure 4.

Recall in that example, $\bar{I}_{au}$ was set to 0.4 after 9 views are processed. However, as the figure shows, in the third iteration of *DiVE-dSwap-Adaptive(0.80)*, an unpruned view V_i was processed that turned out to have an importance score $I(V_i) = 0.6$, which is clearly higher than the upper bound $\bar{I}_{au} = 0.4$. At that point it becomes clear that the the upper bound on importance based on only 9 sample views is inaccurate, and $\bar{I}_{au}$ should be set to the new observed maximum value of 0.6. However, notice that the incorrect value of $\bar{I}_{au} = 0.4$ has already been used for three iterations so far, which means that some high utility views might have been pruned by mistake. As such, simply setting $\bar{I}_{au}$ to 0.6 in the third iteration is clearly insufficient, and previous iterations need to be revisited in light of the new more accurate value of $\bar{I}_{au}$.

As it has been shown in the previous example, setting PI to a small value might lead an inaccurate estimation of I_u and in turn incorrect pruning of high-utility views leading to an overall low utility of recommendations. In fact, the same observation is also true for high value of PI (e.g., $PI = 0.95$), but the impact might be less significant than it is for low PI. This motivated us to extend our adaptive pruning method to rectify the impact of inaccurate estimation of I_u (i.e., *DiVE-dSwap-Rectify(PI)*). Towards this, our proposed rectifying algorithm provides an efficient backtracking mechanism, where the scheme can backtrack to previous iterations to re-evaluate and rectify previous decisions, while at the same time minimizing the processing costs incurred in that backtracking. To achieve this, our rectifying algorithm relies on two important strategies: 1) *Bookkeeping:* to store and keep track of the essential variables in each iteration, and 2) *Caching:* to cache and reuse the importance scores of the views that have already been processed in previous iterations (i.e., avoid recomputing each $I(V_i)$ from scratch) (Algorithm 3 line 7-16).

To understand how our rectifying algorithm works, consider it employed in conjunction with DiVE-dSwap, as shown in Figure 4. As such, for each iteration t, the algorithm stores and maintains the following data structures and variables:

- List L_t: Recall that in each iteration t, all candidate views X are sorted in a list L based on their *setDist* (as discussed in Sec III and Figure 3). To support backtracking, after each iteration t is executed, its corresponding sorted list L_t is saved as part of the bookkeeping mechanism, in case L_t is to be revisited in the future (Algorithm 3 line 13).
- Set S_t: Since in each iteration, a new view might interchange with the view in the current set S, the algorithm stores that updated set, such that each S_t is corresponding to the updated set after executing iteration t (Algorithm 3 line 14).
- Early termination position T_t: This is the index of the view where early termination occurred in list L_t.

Clearly, if backtracking is needed, all the views beyond that position (which have been previously pruned), are reconsidered as they may have a chance to join S under the new estimate of I_u (Algorithm 3 line 15).

The idea underlying our bookkeeping strategy is to keep track and store those essential variables listed above for each iteration so that to efficiently support the cases in which a higher estimated upper bound is found and the upper bound I_{au} is updated. In such cases, our scheme will backtrack to previous iterations, but instead of repeating the processing incurred in each iteration, it can immediately re-use those stored variables. For instance, as shown in Figure 4, a new estimated upper bound is found in the third iteration. Accordingly, the upper bound I_{au} is updated and backtracking takes place to the first iteration, which used an upper bound lower than than the newly estimated one. Particularly, the stored variables for that iteration (i.e., its output set S_t, sorted list L_t, and the early termination position T_t) are all used to re-examine the views that have been pruned in that iteration. Hence, some views which might have previously satisfied the pruning condition $maxO(S \backslash S_j \cup X_i) < O(S)$ may flip to $maxO(S \backslash S_j \cup X_i) > O(S)$ after the upper bound is updated, and are in turn executed to compute their actual importance score (Algorithm 3 line 25-38).

Notice that our rectifying algorithm would need to backtrack every time a new higher upper bound I_{au} is found. Each backtrack might require processing some views that have been initially pruned. However, it is important to notice that a view that has been pruned in one iteration t because it was not promising enough in that iteration, might actually have been processed in a later iteration. To leverage that observation, and to save the cost of processing views while backtracking, our rectifying algorithm utilizes caching to save and maintain the importance score of all processed views. That is, it maintains a cache table of all the views that have been processed in any iteration, together with the importance score of each of those views. Accordingly, when backtracking occurs, and an initially pruned view V_i is to be processed, then that cache table is checked first. If V_i exists in that table, it means that while V_i has been pruned in that revisited iteration, it has actually been processed in another iteration, and its importance score $I(V_i)$ has already been computed and can be re-used directly. As such, a view is processed only if its importance score is not available in the cache table. Clearly, that caching strategy will avoid any duplication in view processing, and guarantees that each view is processed only once, which reduces the number of query execution, while at the same time supports backtracking to improve the quality of recommendation.

V. SHARING-BASED AND HYBRID OPTIMIZATIONS

In this section, we first present our sharing-based optimization (i.e., *Shared-DiVE-dSwap*), which aims to efficiently

Algorithm 3 Rectifying Algorithm

```
Input: Set of views $\mathbb{V}$ and result set size $k$
Output: Result set $S \subseteq \mathbb{V}$, $|S| = k$
1  $S \leftarrow$ set of maximum diversity
2  $X \leftarrow [\mathbb{V} \backslash S]$
3  $PI \leftarrow 0.80$
4  $O(S) \leftarrow$ objective function of $S$ (Eq. 3)
5  $I_{au} \leftarrow AdaptivePruning(PI)$ Alg. 2 (line 13)
6  $t \leftarrow 0$ iteration counter;
7  Function UpdateDataFrameCacheTable(data):
8      if CacheTable not found then
9          create CacheTable
10     else
11         update mode ON
12     end
13     $df_L \leftarrow [t, I_{au}, X_i, S \backslash S_j, setDist]$ Alg. 2 (line 6)
14     $df_S \leftarrow [t, I_{au}, S]$
15     $df_T \leftarrow [t, I_{au}, T]$
16 End Function
17 $maxO(X_i, S \backslash S_j) \leftarrow (1-\lambda) \times \bar{I}_{au} + \lambda \times setDist[X_i, S \backslash S_j]$
18 while improve = True do
19     for $X_i$ in set $X$ do
20         $S' \leftarrow S$
21         for $S_j$ in set $S$ do
22             if $O(S') < O(S \backslash S_j \cup X_i)$ then
23                 $S' \leftarrow S \backslash S_j \cup X_i$
24             end
25             if $maxO(X_i, S \backslash S_j) < O(S)$ then
26                 prune $[X_i, S \backslash S_j]$
27                 data $\leftarrow [t, I_{au}, S, T, [X_i, S \backslash S_j, setDist]]$
28                 UpdateDataFrameCacheTable(data)
29             end
30             if new $I_{au}$ found after more views are
                executed then
31                 backtrack and rectify
32                 update $I_{au}$ to calculate $maxO$
33                 if $maxO(X_i, S \backslash S_j) < O(S)$ then
34                     prune $[X_i, S \backslash S_j]$
35                     data $\leftarrow$
                         $[t, I_{au}, S, T, [X_i, S \backslash S_j, setDist]]$
36                     UpdateDataFrameCacheTable(data)
37                 end
38             end
39         end
40         if $O(S') > O(S)$ then
41             $S \leftarrow S'$
42         end
43     end
44     if $O(S) > O_{current}$ then
45         $O_{current} \leftarrow O(S)$
46         improve $\leftarrow$ True
47     else
48         improve $\leftarrow$ False
49     end
50     $t += 1$
51 end
52 return S
```

reduce the number of query executions when generating view recommendations (Sec. V-A). That optimization is further expanded and integrated with our proposed *DiVE-dSwap-Rectify(PI)*, where combining both techniques leads to maximizing the efficiency of our proposed methods (Sec. V-B).

A. SHARING-BASED OPTIMIZATION

Recall that to automatically recommend interesting views, all possible views are generated first via executing a large number of aggregate queries on all possible combinations of attributes $\mathbb{A}$, measures $\mathbb{M}$ and aggregate functions $\mathbb{F}$. In a naive straightforward approach, each query underlying a view (i.e., the query executed to generate that view) is processed independently on the DBMS. However, notice that those queries scan the same database and different queries still share some common attributes and measures. Hence, in this work, we leverage that similarity and utilize multi-query processing, in which the execution of similar queries is shared so that to reduce the overall query/view processing time.

For instance, consider the Flights dataset [40], which has 7 attributes $\mathbb{A}$, and 4 measures $\mathbb{M}$, and further assume 4 aggregate functions $\mathbb{F}$ are used. Hence, the total number of possible views based on all possible combinations of $\mathbb{A}$, $\mathbb{M}$, and $\mathbb{F}$ is $7 \times 4 \times 4 = 112$. Accordingly, when each view is generated separately, the number of independent queries that are executed on the DBMS is equal to $2 \times 112 = 224$, where for each view one query is executed to generate the target view $V_i(D_Q)$ and another is executed to generate the reference view $V_i(D_R)$ (please refer to Sec. II). This is equivalent to executing the sequence of queries: $(A_1, M_1, F_1), (A_1, M_1, F_2), \ldots, (A_1, M_2, F_1), \ldots,$ $(A_2, M_1, F_1), \ldots, (A_n, M_n, F_n)$ on both D_Q and D_R.

Thus, using some classical multi-query optimization techniques (e.g., [41]), the execution of that sequence of different queries listed above is optimized to reduce query processing. In particular, queries with the same group-by attribute (i.e., A_i) are combined as a single query with multiple aggregations [41]. For instances all queries on A_1 are combined in the re-written query $(A_1, \{F_1(M_1), F_1(M_2)...F_n(M_n)\})$, which is executed as one SQL query on the DBMS. Clearly, using this simple sharing-based multi-query optimization, reduces the number of queries on the Flights dataset from 224 to only $2 \times 7 = 14$ queries.

At this point, it might seem that this sharing-based optimization *Shared-DiVE-dSwap* presented above is in conflict with our pruning strategies *DiVE-dSwap-Rectify(PI)* proposed in the previous sections.

Both the sharing-based optimization *Shared-DiVE-dSwap* presented above, as well as our pruning strategies *DiVE-dSwap-Rectify(PI)* proposed in the previous sections achieve significant reductions in query processing time. However, those reductions are achieved through different optimizations (i.e., pruning vs. sharing), and are orthogonal to each other. Particularly, on the one hand, *Shared-DiVE-dSwap* favors generating all possible views by executing all their underlying

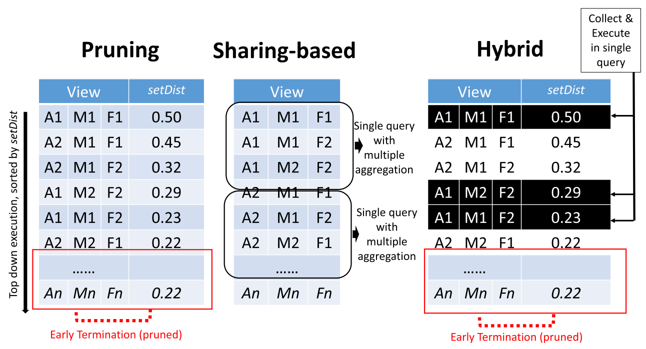


FIGURE 5. Adaptive pruning with rectifying ***DiVE-dSwap-Rectify(PI)*** vs. Sharing-based ***Shared-DiVE-dSwap*** vs. Hybrid optimization ***Shared-DiVE-dSwap-Rectify(PI)*** in a simple illustration.

queries at the same time. On the other hand, our *DiVE-dSwap-Rectify(PI)* favors eliminating the generation of as many views as possible by leveraging the properties of diversity and the upper bound on importance. This is achieved by: 1) ordering/ranking the queries according to their utility scores, and 2) pruning those queries of low-utility.

Hence, when sharing the execution of the set of queries $(A_1, \{F_1(M_1), F_1(M_2)...F_n(M_n)\})$, the entire set is executed as one single query to leverage shared processing. To the contrary, when applying our pruning methods, the different queries in that set will be considered in random non-sequential order based on their utility scores, and many of them are likely to be pruned. That is, each query in the set is executed on-demand to maximize pruning and reduce the number of query executions. In Sec VII, we compare the performance of *Shared-DiVE-dSwap* vs. *DiVE-dSwap-Rectify(PI)*, while in the next section we propose a new scheme, which combines the benefits of both pruning and shared optimization, *Shared-DiVE-dSwap-Rectify(PI)*.

B. HYBRID OPTIMIZATION

To explain our hybrid optimization methods (i.e., *Shared-DiVE-dSwap-Rectify(PI)*), consider Figure 5, which illustrates a visual comparison between three techniques, namely: adaptive pruning DiVE-dSwap-Adaptive(PI), sharing-based Shared-DiVE-dSwap, and hybrid (pruning + sharing-based) Shared-DiVE-dSwap-Rectify(PI). In particular, the figure expands on our earlier Figure 5 and shows the list of queries to be executed during one iteration of our DiVE-dSwap. The figure also shows that under the pruning techniques (left-hand side), those queries are sorted by *setDist*, where the top queries with high value for *setDist* are executed until early termination is reached, as explained in Sec. III-B.

Meanwhile, Shared-DiVE-dSwap combines queries with the same group-by attribute into a single query with multiple aggregations. For instance, as Figure 5 shows, the three queries that perform group-by on attribute A_1 are combined into one shared query, and the same for the other three queries on attribute A_2. Hence, Shared-DiVE-dSwap replaces the execution of those 6 highlighted individual queries with the execution of only 2 shared queries.

Now turning our attention to our new hybrid method Shared-DiVE-dSwap-Rectify(PI), Figure 5 shows that, in general, it follows the same logic required for basic query pruning, where queries as are sorted by *setDist*. However, differently from the basic pruning approach, in Shared-DiVE-dSwap-Rectify(PI), whenever a query is selected for execution (i.e., unpruned query), it is executed together with other queries that have the same group-by attribute. For example, as the figure shows, when the top query in the list is selected for execution, then it is combined together with other queries with a group-by attribute A_1 and are all processed as one single shared query with multiple aggregations.

As such, Shared-DiVE-dSwap-Rectify(PI) combines the best of both worlds! That is, instead of executing all groups of shared queries, it selectively executes those shared groups that contain queries with high priority, according to our pruning methods. This provides two simultaneous benefits: 1) eliminating the processing of some queries when early termination is reached, and 2) reducing the processing time of unpruned queries by utilizing shared processing.

Clearly, the amount of pruning achieved by this hybrid method is expected to be less than that provided by a pure pruning method. That is, the number of queries that are eliminated without processing is reduced under the hybrid method. This is because some of the queries that would have been eliminated under the pruning method, might now be executed because they are shared with some high-priority unpruned query. However, processing those queries would come at a minimal cost because of the shared processing.

From the discussion above, it should be clear that the number of queries which are combined into a single shared group-by query (i.e., sharing factor) affects the performance of our hybrid optimization. In particular, consider again the top query in the sorted list shown in Figure 5, which performs a group-by on attribute A_1. Clearly, for that query, the maximum sharing factor or number of other queries that it can be combined with for shared execution is equal to: $\mathbb{M} \times \mathbb{F}$. In that extreme case, our hybrid method will simply converge to a pure shared-based optimization since all queries with the same group-by attribute are combined and executed as a single query with multiple aggregations. That is, there will be minimum potential for eliminating low-priority queries based on our pruning methods. To the contrary, if each query on the list is executed as a single query without employing any shared processing, then our hybrid approach will be exactly equivalent to our pruning methods discussed in the previous section. That is, the reductions in query processing time will only come from pruning without any additional savings from shared processing. Hence, we introduce a parameter called sharing factor (β), which acts as a knob that controls the amount of desired shared processing. Particularly, β is tuned in the range $0\% \leq \beta \leq 100\%$. For instance, a higher value of β results in maximazing the amount of shared-based processing, whereas a lower value of β maximizes the amount of adaptive pruning. In Sec. VII, we study that trade-off and show the impact of the sharing factor parameter on the performance of our hybrid approach.

TABLE 1. Parameters testbed in the experiments.

Parameter	Range (**default**)
datasets	Heart disease, Flights, Diabetes
diversity weight ratio	$\mathbf{3}(A) : \mathbf{2}(M) : \mathbf{1}(F)$
tradeoff weight λ	0.0, 0.2, 0.4, **0.5**, 0.6, 0.8, 1.0
result set (size of k)	**5**, 15, 25, 35
diversity function	*MaxSum*, *MaxMin*
prediction interval %	70, 80 , 85, 90, 95, 97
aggregate functions	Max Min Avg Sum
sharing factor (β)	0, 20, 40, **50**, 60, 80, 100

VI. EXPERIMENTAL TESTBED

In this section, we present the extensive experimental evaluation of our DiVE scheme on real data sets. Table 1 summarizes the parameters used in our evaluation (default values are in bold).

We conducted our experiments over the following datasets: 1) Heart Disease Dataset [42]: This dataset is comprised of 8 dimensional attributes and 6 measure attributes, using four aggregate functions, resulting in a total of $2 \times 8 \times 6 \times 4 = 384$ possible views, and 2) Flights Dataset [40]: This dataset is comprised of 7 dimensional attributes and 4 measure attributes for a total of $2 \times 7 \times 4 \times 4 = 224$ possible views. While its dimensionality is lower than the heart disease data, it is a relatively large dataset of almost one million tuples, which helps in evaluating the incurred query processing time. (3) The Diabetes 130 US hospital dataset [17] consists of 14 dimensional attributes, 13 measures and 100 thousand tuples, for a total of $2 \times 14 \times 13 \times 4 = 1456$ possible visualizations.

For each experiment, the performance measures are averaged over a query workload of ten random queries submitted to select ten different subsets. The default value for k in our evaluation is 10, whereas the diversity weight ratio is $3(A) : 2(M) : 1(F)$. That is, dimensional attributes have the highest similarity weight in the Jaccard computation, followed by measure attributes, and the least weight is given to aggregate functions (please see Section II). The default trade-off weight λ to balance between importance and diversity is set to 0.5. Finally, the default diversity function is *MaxSum*, however, we also present our results on *MaxMin* (as it has been presented in Section II).

In terms of evaluated methods, as mentioned earlier, our work in [11] shows that DiVE-dSwap provides the best performance in terms of both effectiveness and efficiency compared to existing solution. However, for the sake of completeness, in this work we re-evaluate DiVE-dSwap and its proposed extension against the baselines discussed in [11]. Particularly, in terms of diversity, we employ the classical Greedy Construction algorithm, which has been shown to maximize diversity within reasonable bounds compared to the optimal solution (e.g., [30], [33], [43]). In this work, we refer to that baseline as Greedy-Diversity. Similarly, in terms of importance, we adopt the work on SeeDB for recommending the top-k views with the highest deviation [7]. Particularly, in that method, all possible target and reference views are generated by executing their underlying queries, then the list of views is linearly scanned to recommend the top-k for which the target view shows high deviation from its corresponding reference view (denoted as Linear-Importance in this work).

Those baseline, together with our proposed DiVE-dSwap methods are summarized as follows:

- *Greedy-Diversity*: select the top-k views that maximize the total diversity of set S.
- *Linear-Importance*: select the top-k views with the highest deviation value (the importance score).
- *DiVE-dSwap*: a local search algorithm starts out with a complete initial solution and then attempts to find a better solution in the neighborhood of that initial one, where set S is initiated with top-k most diverse views.
- *DiVE-dSwap-Static*: DiVE-dSwap with the static pruning (theoretical upper bound)
- *DiVE-dSwap-Adaptive(PI)*: DiVE-dSwap with adaptive pruning methods, where PI is based on *non-parametric predictive intervals*
- *DiVE-dSwap-Rectify(PI)*: DiVE-dSwap with adaptive pruning methods and rectifying, PI is *non-parametric predictive intervals* and R is rectifying.
- *Shared-DiVE-dSwap*: DiVE-dSwap with sharing-based optimization
- *Shared-DiVE-dSwap-Rectify(PI)*: hybrid method, the combination of adaptive pruning with rectifying method DiVE-dSwap-Rectify(PI) and sharing-based optimization Shared-DiVE-dSwap

VII. EXPERIMENTAL EVALUATION

A. IMPACT OF λ ON EFFECTIVENESS

Figures 6 and 7 illustrate the impact of varying λ on the performance of each scheme in terms of effectiveness (i.e., the value of the objective function $O(S)$). As λ decreases, Linear-Importance consistently achieves the highest $O(S)$. Conversely, as λ approaches 1, Greedy-Diversity attains the highest $O(S)$. Consequently, there exists a crossover point between the two schemes. Nevertheless, DiVE-dSwap demonstrates a stable performance for all values of λ, surpassing both Linear-Importance and Greedy-Diversity.

B. IMPACT OF k ON EFFECTIVENESS

Figures 8 and 9 illustrate the $O(S)$ values for various schemes as the number of recommended views k varies from 5 to 35 for the Heart disease and Flight datasets, respectively. Overall, $O(S)$ decreases as k increases for all schemes. This is due to the decrease in both the average importance score and diversity of a set S as new views are added. The views

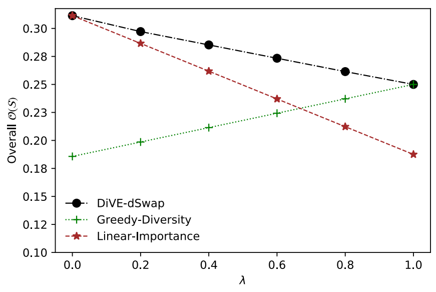


FIGURE 6. The impact of λ on O over heart disease dataset.

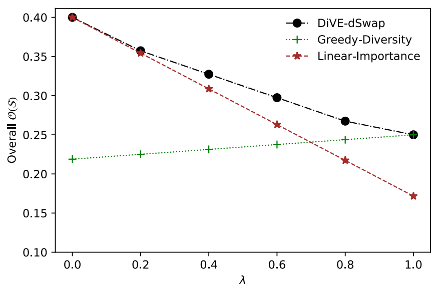


FIGURE 7. The impact of λ on O over flight dataset.

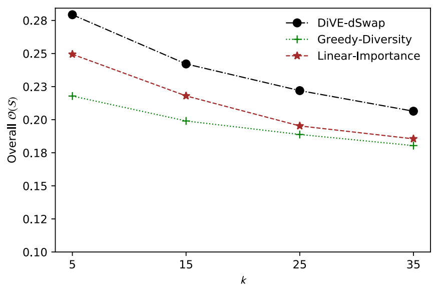


FIGURE 8. The impact of k on O over heart disease dataset.

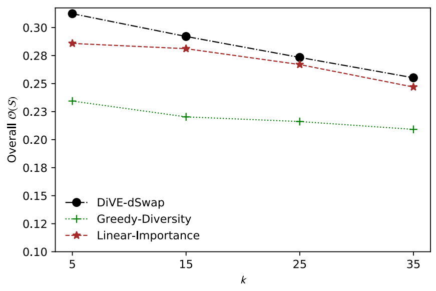


FIGURE 9. The impact of k on O over flight dataset.

added earlier to S have a higher importance score than those added later. Similarly, the diversity function shows a diminishing marginal gain trend as the size of set S increases. An important observation is that our DiVE-dSwap approach consistently has higher overall objective function values compared to the two extreme baseline approaches for all values of k.

C. IMPACT OF DIVERSIFICATION FUNCTIONS ON DiVE-dSwap

Figures 10 and 11 illustrate the impact of varying λ on the performance of DiVE-dSwap as measured by $O(S)$ using

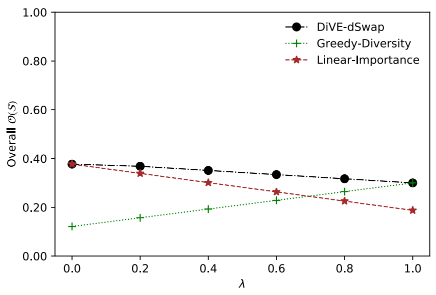


FIGURE 10. The impact of λ on *O* over diabetes dataset using Max-SUM diversity function.

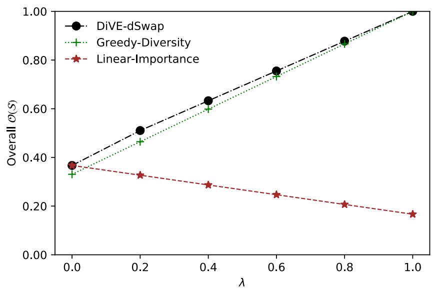


FIGURE 11. The impact of λ on *O* over diabetes dataset using Max-MIN diversity function.

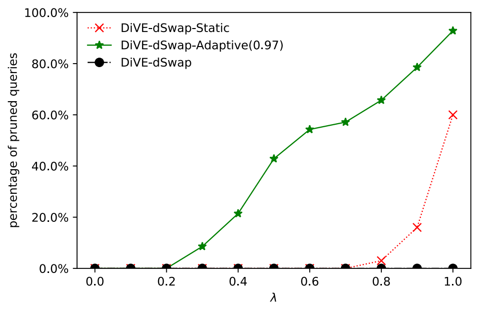


FIGURE 12. DiVE-dSwap static pruning vs adaptive pruning on heart disease dataset.

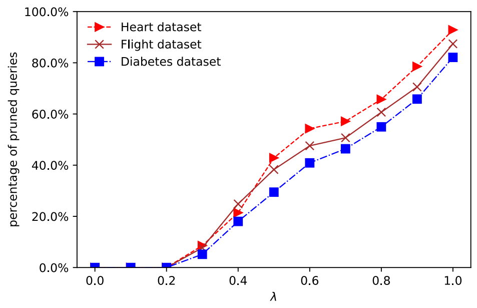


FIGURE 13. DiVE-dSwap-Adaptive(0.97) on three different datasets.

different diversity functions on the Diabetes dataset. Recall that our previous work [11] focused on the Max-SUM diversity function, wheres in this paper, we also study the impact of adopting the Max-MIN alternative diversity function. Also recall that the goal of Max-SUM is to maximize the average pairwise distance among the objects in a set, while the goal of Max-MIN is to maximize the minimum pairwise distance among those objects. For instance, suppose a set S comprises three views, where all views have different values for A, M, and F. The maximum distance between any two views in set S is 1.0. Using the Max-SUM diversity function, the diversity score of set S is the total sum of all distances divided by $k * (k-1)$, i.e., $f(S, D) = \frac{(1+1+1)}{3(3-1)} = 0.5$. In contrast, using the Max-MIN function, the diversity score is $f(S, D) = 1.0$ due to the views having different A, M, and F. Consistently with that observation, Figures 10 and 11 show that, in general, the overall value of $O(S)$ tends to be higher for all values of λ when using Max-MIN vs. Max-SUM. That increase in $O(S)$ is even further magnified at higher values of λ, in which the Max-MIN high diversity score tends to dominate the overall $O(S)$ function. The figures also show that for both the Max-SUM and Max-MIN diversity functions, DiVE-dSwap automatically adapts to the value of λ and outperforms the baseline methods.

D. IMPACT OF λ ON THE PERCENTAGE OF PRUNED QUERIES

In Figures 12 and 13, we evaluate the performance of our proposed pruning techniques in terms of the number of pruned queries. We compare DiVE-dSwap-Static and DiVE-dSwap-Adaptive(0.97) over different values of λ, as we execute them on the Heart disease dataset. In this comparison, we also include DiVE-dSwap (i.e., no pruning) to serve as a performance yardstick. Overall, the results show that DiVE-dSwap-Adaptive(0.97) has better performance in terms of the number of pruned views compared to DiVE-dSwap-Static. We note that the number of pruned queries increases significantly for higher values of λ. Particularly, at higher value of λ, $O(S)$ is dominated by the diversity score, thereby increasing the pruning power. In comparison to DiVE-dSwap-Adaptive, recall that DiVE-dSwap-Static utilizes the theoretical upper bound of the importance score, I_u. Since that theoretical bound typically overestimates the actual bound on the importance score, DiVE-dSwap-Static provides limited pruning power, and is only able to prune some queries for values of λ higher than 0.8, as shown in Figure 12.

E. IMPACT OF *PI* ON DiVE-dSwap-ADAPTIVE

In this experiment, we evaluate the performance of the DiVE-dSwap-Adaptive adaptive pruning technique under varying values of *PI* and λ. As depicted in Figure 14, low *PI* results in a high number of pruned queries. For example, DiVE-dSwap-Adaptive(0.70) has the highest number of pruned queries, with 48% of queries being pruned when the default λ value of 0.5 is used. Note that in this comparison, we have introduced a new benchmark method that we annotate as DiVE-dSwap(Hypothetical). In that method, we assume that the actual upper bound on importance is accurately known in advance. As expected, under that hypothetical assumption, DiVE-dSwap(Hypothetical) provides significant pruning power (Figure 14), at no loss in effectiveness (Figure 15). Particularly, Figure 15 shows the loss of

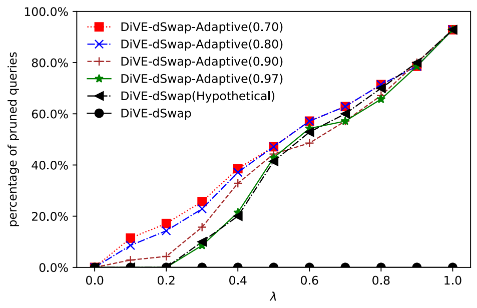


FIGURE 14. **DiVE-dSwap-Adaptive *PI* values vs. ratio of pruned queries.**

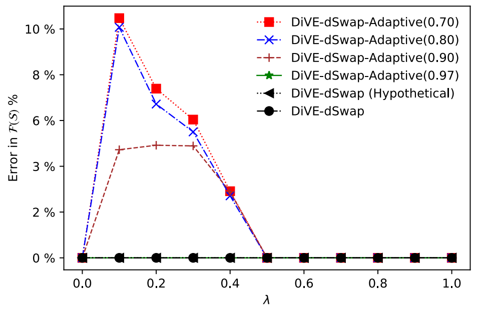


FIGURE 15. ***PI* values vs. Error on *O*(*S*) *PI* values vs. Error in *O*(*S*).**

DiVE-dSwap-Adaptive in $O(S)$ in comparison to the $O(S)$ achieved by DiVE-dSwap(Hypothetical). As the figure shows, that loss is 0% for $PI = 0.97$, which is equivalent to using a large sample size for estimating the importance upper bound, and in turn achieving higher accuracy (note that in the figure, DiVE-dSwap-Adaptive(0.97) coincides with both DiVE-dSwap and DiVE-dSwap(Hypothetical)). As the sample size gets smaller (i.e., lower values of PI), that accuracy in estimating the upper bound is decreased, leading to some loss in the achieved effectiveness. For instance, as Figure 15 shows, the loss in effectiveness might go up to 10% for $PI = 0.70$. We also note that the loss in $O(S)$ decreases as λ increases, as the impact of importance score becomes smaller in the hybrid objective function. Maximizing the effectiveness, while at the same time minimizing the number of processed queries, is the motivation for our newly proposed rectifying method, which is evaluated next.

F. IMPACT OF RECTIFYING

Figure 16 compares the performance of DiVE-dSwap-Rectify(PI) and DiVE-dSwap-Adaptive(PI), where DiVE-dSwap-Rectify(PI) employs both adaptive pruning and rectifying, whereas DiVE-dSwap-Adaptive(PI) employs only adaptive pruning. As shown in the figure, for both DiVE-dSwap-Rectify(PI) and DiVE-dSwap-Adaptive(PI), a lower *PI* results in a higher number of pruned queries. For example, when $PI = 0.50$, more queries are pruned compared to when $PI = 0.90$. However, for the same value of *PI*, DiVE-dSwap-Rectify(PI) typically incurs a lower error rate than DiVE-dSwap-Adaptive(PI). For example, when $PI = 0.90$, DiVE-dSwap-Adaptive(0.90) prunes the

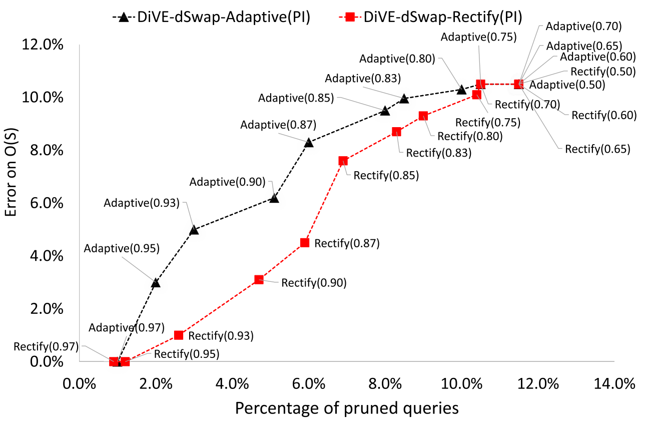


FIGURE 16. **Comparison between DiVE-dSwap-Rectify(PI) and DiVE-dSwap-Adaptive(PI) in terms of the percentage of pruned queries and the error in *O*(*S*).**

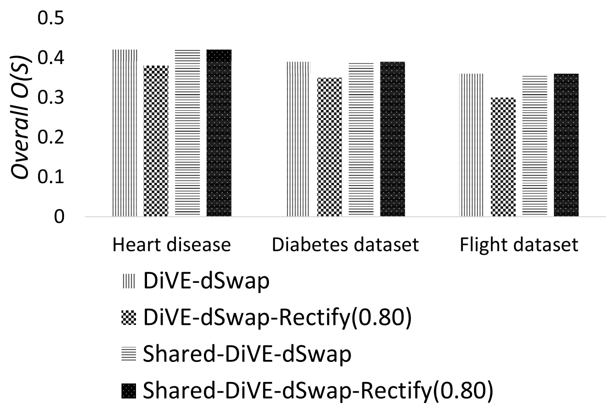


FIGURE 17. **The effectiveness of sharing-based optimization over all datasets.**

processing of up to 6% of the queries, but incurs a 6% error in $O(S)$. In contrast, for the same value of PI, DiVE-dSwap-Rectify(0.90) prunes up to 5% of the queries, while incurring only 3% error, making it more effective. Additionally, while achieving the same error rate of 3%, DiVE-dSwap-Rectify(0.90) can prune up to 5% of the queries, while DiVE-dSwap-Adaptive(0.95) can only prune 2% of those queries to achieve the same small error rate of 3%.

G. PERFORMANCE OF HYBRID OPTIMIZATION

In this experiment, we compare the performance of four optimization methods in terms of both effectiveness and efficiency: 1) *DiVE-dSwap*, 2) *DiVE-dSwap-Rectify(0.80)*, which combines adaptive pruning and rectifying, 3) *Shared-DiVE-dSwap*, which employs sharing-based optimization, and 4) *Shared-DiVE-dSwap-Rectify(0.80)*, which combines adaptive pruning with rectifying and sharing-based optimization, and we denote as hybrid optimization. We note that in this experiment, *DiVE-dSwap* and *Shared-DiVE-dSwap* are used as baselines since they execute all query views to generate recommendations without employing any pruning. Figure 18 shows that our hybrid approach, Shared-DiVE-dSwap-Rectify(0.80) with $\beta = 50\%$ provides the

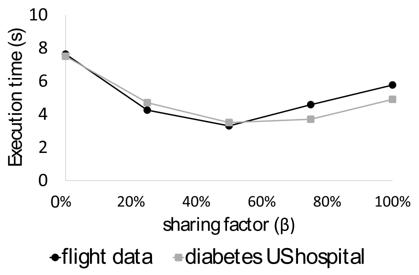


FIGURE 18. The impact of β on our hybrid optimization efficiency.

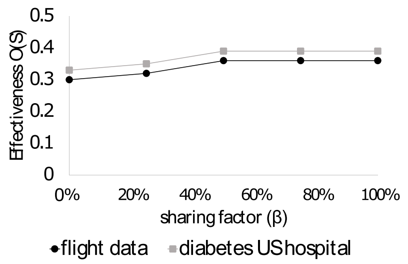


FIGURE 19. The impact of β on our hybrid optimization effectiveness.

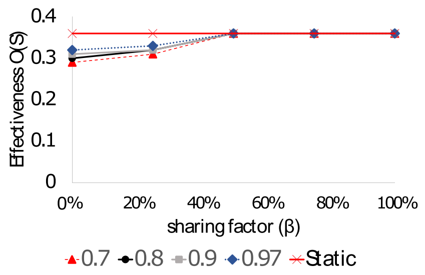


FIGURE 20. The Impact of β on our hybrid optimization effectiveness - Shared-DiVE-dSwap-Rectify(0.80) using different *PI*s.

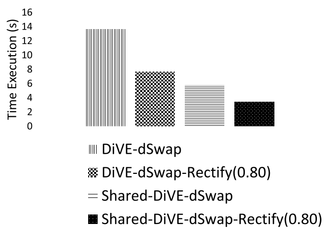


FIGURE 21. The efficiency of sharing-based optimization over the flights dataset.

minimum query execution time over the Flights dataset. That reduction in execution time is achieved while maintaining high effectiveness, as shown in Figure 17, which depicts the quality of recommendations achieved by all methods for all our employed datasets.

H. THE IMPACT OF SHARING FACTOR β ON HYBRID OPTIMIZATION EFFICIENCY

As described in Sec.V-B, our proposed hybrid method combines both pruning and sharing-based query processing. That is, low-utility queries are pruned, whereas unpruned queries with the same group-by attribute are combined and processed as a single query with multiple aggregations. Figure 18 illustrates the impact of the sharing factor β on the performance of our hybrid Shared-DiVE-dSwap-Rectify(0.80) in terms of efficiency. As the figure shows, the query execution time starts high, then it decreases with increasing β, until a point where it starts increasing again (U-shape). That is because at low β values, there is minimum shared processing, and the only reduction in execution time comes from pruning. Oppositely, at high β values, many queries are combined together, which minimizes the chance of pruning, and the reduction in execution time is achieved only from the shared query processing. As the figure shows, for this experiment the best balance between shared execution and pruning is achieved around $\beta = 50\%$, which provided the minimum execution time.

I. IMPACT OF SHARING FACTOR β ON HYBRID OPTIMIZATION EFFECTIVENESS

Figure 19 shows the impact of the sharing factor β on the performance of Shared-DiVE-dSwap-Rectify(0.80) in terms of effectiveness. The results indicate that that effectiveness increases as β increases. That is, the more queries are executed because of high-degree of shared processing, the more accurate is the estimated upper bound on view importance $\bar{I}_{au}$, and in turn higher accuracy in pruning low-importance queries. However, we note that in this experiment the effectiveness of Shared-DiVE-dSwap-Rectify(0.80) keeps increasing up until $\beta = 50\%$, then it levels off and stays constant. In other words, setting $\beta = 50\%$ lead to executing a large enough sample of queries to discover the actual upper bound on importance I_{au}. This is further investigated in Figure 20, in which we examine the interplay between β and different values of PI. As expected, the figure shows that higher values of PI lead to higher effectiveness. However, the figure also shows that by increasing β, all versions of Shared-DiVE-dSwap-Rectify(PI) reach the maximum effectiveness, regardless of their PI value.

VIII. RELATED WORK

A. INSIGHT RECOMMENDATION SYSTEMS

Recent years have seen the introduction of visual insight recommendation systems due to the increasing user demand for automated insight discovery (e.g., [4], [5], [6], [7], [8], [9], [10], [12], [13], [21], [45], [46]). Generally, visual insight recommendation systems can be divided into the following three categories:

- Task Driven Recommendation: focuses on recommending interesting visualizations that facilitate particular user objectives and tasks such as finding missing values, outliers, or specific patterns.
- Feedback Driven Recommendation: focuses on enabling the discovery of personalized interesting visualizations by utilizing the user's feedback as inputs to the recommendation systems.

TABLE 2. A summary of data-driven visualization recommendation systems.

Approach	Ranking Metric	Optimization
SeeDB [7]	Importance score (deviation-based approach)	Utilizes multi-armed bandits to prune low-quality visualizations and a sharing-based technique for multi-query processing
DeepEye [13]	Hybrid (chart type, visualized data correlations & distribution)	Pruning of the search space based on a directed graph (DAG) representing the partially ordered set of visualizations (i.e., a Hasse diagram)
TopkAttr [21]	Importance score (deviation-based approach)	Introduces adaptive querying solution to estimate the importance score of visualizations. It divides the dataset into partitions, calculates the importance score for each partition, and then aggregates the overall importance score
QuickInsights [3]	Developed "interestingness" scoring for 12 types of insights	Utilizes pruning strategy to eliminate any visualization with an impact smaller than a given threshold, with a caching mechanism that utilize prefetched results
VizRec [44]	Importance score (deviation-based approach)	Focuses on designing mechanisms to avoid making false discoveries. While it emphasizes statistical guarantees, it lacks optimizations for improved efficiency
MuVE [8]	Hybrid (importance score + binning properties)	Utilizes a pruning strategy which is designed based on the multi-objective utility function that supports binned visualizations
Zenvisage [22]	Score of similarity to some given pattern	Introduces the ZQL query optimizer, which compiles ZQL queries into a directed acyclic graph (DAG) of operations on collections of visualizations It provides substantial improvements over DBMS multi-query optimization.
DiVE (this work)	Hybrid (importance score + diversity score)	Adaptive pruning of the views search space, and shared-based query processing

- Data Driven Recommendation: focuses on enabling the discovery of interesting visualizations from large volumes of data based on data characteristics such as summaries, distribution, correlation, historical data, corpus, labeled data, or data collected offline. Machine learning methods, either supervised or unsupervised learning, are often used in this approach.

It is also possible for a visual insight recommendation system to combine elements of two or more of these categories, such as a task-driven and feedback-driven system or a task-driven and data-driven system. A detailed explanation of these categories will be provided in the next section.

- **Task Driven Recommendation**: The Profiler system [9] detects anomalies in data by utilizing mutual information metrics and then recommends visualizations (termed "anomaly insights") to users to aid in resolving data quality issues. Another example of a task-driven recommendation system is TaskVis [47]. TaskVis categorizes visualization tasks into 18 analytic tasks, based on a survey of industry and academia. These tasks include analysis of distributions, clustering, comparisons, changes over time, identifying anomalies and trends, among others. TaskVis recommends visualizations based on the user's specified task and interests. The user inputs their task and interests, and TaskVis generates the recommended visualizations.
- **Feedback Driven Recommendation**: The VizRec system [48] employs three common methods in recommender systems to recommend visualizations to users: 1) Collaborative Filtering (CF), in which recommendations are constructed based on the user's past ratings of visualizations and the recommended visualizations are similar to those the user has previously rated. 2) Content-based Filtering (CBF), which is used to address the "cold start" problem that arises when new users do not have historical rating data. 3) Hybrid Filtering, a combination of CF and CBF that provides benefits to users, particularly when their interests change. Another example of a feedback-driven recommendation system is AIDE [49]. It employs a human-in-the-loop approach to learn user interests based on their feedback. AIDE can guide users through the data space they wish to explore and refine its predictions based on the user's feedback.
- **Data Driven Recommendation**: DeepEye [13], [50] is a visual insight recommendation system that employs a supervised machine learning approach. The system aims to capture human perception by understanding existing examples using ML-based classifier techniques. Specifically, DeepEye trains a binary classifier to decide whether a particular visualization is good or bad. To train the machine learning models, DeepEye utilizes two types of labeled datasets: 1) labeled visualizations as good or bad with respect to all possible candidate visualizations; 2) labeled visualizations as good, from two compared visualizations. Additionally, the system employs a supervised learning-to-rank model to rank visualizations. A more comprehensive explanation of this approach can be found in [51]. SeeDB [7] is one of the first visual insight recommendation system which recommends top-k aggregate visualizations based on data-driven deviation-based approach. A user study is conducted in that work, and it provides strong evidence that a deviation-based formulation of utility is able to provide analysts with interesting visualizations. TopkAtrr [21] leverages a deviation-based approach to recommend attributes and it guarantees the correctness of its recommendations. Similarly, VizRec [44] employs a deviation-based approach, but with a focus on eliminating the risk of discovering false insights or misleading visual recommendations through the use of classical statistical testing. QuickInsights [3] defines its interestingness as data that is trending upwards or downwards rapidly and consistently. In addition,

QuickInsights supports multiple types of insights for a more comprehensive analysis. Zenvisage [22] also employs a deviation-based approach, but defines interestingness as visualizations that are similar to the user's desired pattern. To do this, Zenvisage provides an empty chart for the user to draw their desired pattern, and subsequently recommends visualizations that have similar patterns. Other works that leverage a deviation-based approach include MuVE [15], which addresses binning problems in visualization recommendation systems. MuVE proposes a hybrid multi-objective utility function that captures the impact of numerical dimension attributes in generating visualizations that are: 1) interesting (deviation-based), 2) usable (not too cluttered), and 3) accurate. DiVE [11] proposes hybrid utility functions that capture both the importance (deviation-based) and the diversity of the recommended insights.

Table 2 presents a summary of current approaches for data-driven visualization recommendation. For each approach, we summarize the metrics used for the ranking of the recommended visualizations. In addition, since efficiency has been a main goal of this work, we also briefly discuss the data and query processing optimizations proposed by each of the presented approaches. For more details, we refer the reader to some recent surveys on this topic, including [51], [52].

B. THE IMPORTANCE OF DIVERSIFIED RECOMMENDATION RESULTS

Diversification is a commonly used feature in recommendation systems to enhance user satisfaction and improve their experience (e.g., [16], [26], [27], [28], [29], [30], [31], [32]). The goal of recommending diversified results is to reduce the risk of redundancy and increase the coverage of recommendations. A recommender system without diversification may return results that are too similar to meet users' needs [29]. In contrast, a recommender system with diversification returns a set of representative results that are not only relevant to the user query, but also diverse. In the context of visual insight recommendation, the absence of diversification may lead to the recommendation of redundant visualizations, limiting the scope of possible insights for the user.

There are various types of diversification methods proposed in the literature. These types of diversification can be classified into one of the following categories: 1) Content-based diversification, which aims to select results that are dissimilar to each other (e.g., [11], [30], [53]); 2) Novelty-based diversification, which aims to select results that contain new information when compared to what was previously presented to the user [27]; and 3) Semantic-based diversification, which aims to select results that belong to different categories or topics [54]. Essentially, these various types of diversification have different focuses. Content-based diversification emphasizes the dissimilarity of the recommended results, while novelty-based diversification focuses on recommending new information. Semantic-based diversification aims to provide a comprehensive understanding of user needs by resolving ambiguous queries and returning results from different categories or topics. However, in this work, we focus on content-based diversification to increase the overall diversity of the recommendation results and to avoid redundancy.

There are numerous algorithms that return diversified top-k results, however, most of them are generally based on these two popular algorithms: Greedy MMR [55], and SWAP [30], [33]. The Maximal Marginal Relevance (MMR) algorithm [55] balances between the relevance score and the diversity score of the recommended set S by introducing a λ coefficient. The algorithm is greedy in nature, growing the size of the top-k set by adding items one by one while considering the relevance of the item and its diversity with previously selected items. The SWAP algorithm [30], [33] is a local search algorithm that starts by sorting the items by relevance and initializing the top-k result set S with the k-items with the highest relevance score. In each iteration, it scans unselected items and swaps one item with a candidate item if the item has a higher contribution to diversity. The algorithm terminates when the unselected items have been fully scanned. These algorithms have complex steps for computing the pairwise diversity among all items to generate diversified top-k results. For large databases, this repetitive computation is computationally expensive. The work in [56] develops a generic framework for efficient computation of top-k diverse results, consisting of a function called *DivGetBatch()* that replaces repeated pairwise comparisons of diversity scores of items with pairwise comparisons of "aggregate" diversity scores of a group of items and a structure called *I-tree*, a hierarchical complete tree-like structure that divides a database containing N items into multiple groups of items. That framework ensures up to a 24× speedup in computation of top-k diverse results.

It is important to note that various diversity functions have been employed in the literature to achieve diversity when selecting a set of objects, including the MaxMin and MaxSum functions. The goal of the MaxSum function is to maximize the average pairwise distance between the objects in the recommendation results, whereas the goal of the MaxMin function is to maximize the minimum pairwise distance between the recommendation results. In this work, we consider both notions of diversity functions (MaxMin and MaxSum) and both the Greedy and Swap algorithms when implementing diversification to generate diversified top-k visual insights.

IX. LIMITATIONS AND FUTURE WORK

Our proposed DiVE scheme adopts a data-driven approach in recommending interesting visualizations, which clearly has its advantages as well as its disadvantages. One general advantage of the data-driven approach is its ability to reduce biases that might arise from human judgment, as recommendations are based on objective data-driven metrics. Meanwhile, a disadvantage is the lack of personalization, as data-driven recommendations may not be tailored to individual user preferences and needs. However, DiVE takes one step towards that personalization by introducing diversification so that the recommended visualizations provide a wider coverage of the possible insights to be discovered. That is, in addition to the data-driven notion of importance, providing a diversified set of visualizations would cater for different personalized analytical needs.

However, DiVE still provides limited user-driven personalization, which we plan to address in our future work. For instance, currently, we assume that the user is able to provide a pre-specified weight to balance the tradeoff between importance and diversity (i.e., the λ parameter in Eq.3). In the future, we plan to utilize active learning methods in order to automatically learn the user preference for that tradeoff (e.g., [49]), and in turn adaptively set the corresponding weight based on the user's feedback. Similarly, we will also investigate learning the user preference for tuning our context-driven similarity measure. Particularly, we will learn what makes two visualizations (dis)similar from the user's perspective and accordingly adapt the weights in our utilized Jaccard distance measure (please see Eq.2). Aside from the deviation-based metric, we plan to expand our multi-objective hybrid function to integrate diversity with other data-driven measures of importance [4], [57] (e.g., correlation, skewness in distribution, etc.). Finally, as we already exploited in this work the mathematical characteristics of the deviation-based metric to optimize the incurred query processing (i.e., query pruning based on the metric bounds), in the future, we will also investigate novel optimization methods that suit those new multi-objective functions.

X. CONCLUSION

In this work, we propose several optimization techniques for incorporating diversification in the process of recommending insightful visualizations. Central to our proposed techniques is a hybrid utility function, which combines the importance of the recommended visualizations, together with the diversity of those recommendations. The main idea underlying our proposed techniques is to prune the processing of a large number of low-utility visualizations, while sharing the processing of the remaining unpruned and, potentially, high-utility ones. This allows for minimizing the query processing time incurred during the recommendation process, while at the same time maximizing the quality of those recommendations, as it has been shown through our extensive experimental evaluation.

ACKNOWLEDGMENT

(Mohamed A. Sharaf, Rischan Mafrur, and Guido Zuccon contributed equally to this work.)

REFERENCES

[1] Tableau Public. *Tableau Helps People See and Understand Data*. [Online]. Available: https://public.tableau.com/s/

[2] Power BI. *Microsoft Power BI: Interactive Data Visualization BI Tools*. [Online]. Available: https://powerbi.microsoft.com/en-us/

[3] R. Ding, S. Han, Y. Xu, H. Zhang, and D. Zhang, "QuickInsights: Quick and automatic discovery of insights from multi-dimensional data," in *Proc. Int. Conf. Manage. Data*, Jun. 2019, pp. 317–332.

[4] D. Emiralp, J. P. Haas, S. Parthasarathy, and T. Pedapati, "Foresight: Recommending visual insights," in *Proc. PVLDB*, 2017, pp. 1–4.

[5] A. Key, B. Howe, D. Perry, and C. Aragon, "VizDeck: Self-organizing dashboards for visual analytics," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, May 2012, pp. 681–684.

[6] M. Vartak, S. Madden, G. A. Parameswaran, and N. Polyzotis, "SEEDB: Automatically generating query visualizations," in *Proc. PVLDB*, 2014, pp. 1–4.

[7] M. Vartak, S. Rahman, S. Madden, G. A. Parameswaran, and N. Polyzotis, "SEEDB: Efficient data-driven visualization recommendations to support visual analytics," in *Proc. PVLDB*, 2015, pp. 1–12.

[8] H. Ehsan, M. A. Sharaf, and P. K. Chrysanthis, "MuVE: Efficient multi-objective view recommendation for visual data exploration," in *Proc. IEEE 32nd Int. Conf. Data Eng. (ICDE)*, May 2016, pp. 731–742.

[9] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer, "Profiler: Integrated statistical analysis and visualization for data quality assessment," in *Proc. Int. Work. Conf. Adv. Vis. Interface*, May 2012, pp. 547–554.

[10] J. Seo and H. Gordish-Dressman, "Exploratory data analysis with categorical variables: An improved rank-by-feature framework and a case study," *Int. J. Hum.-Comput. Interact.*, vol. 23, no. 3, pp. 287–314, Dec. 2007.

[11] R. Mafrur, M. A. Sharaf, and H. A. Khan, "DiVE: Diversifying view recommendation for visual data exploration," in *Proc. 27th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2018, pp. 1123–1132.

[12] B. Tang, S. Han, M. L. Yiu, R. Ding, and D. Zhang, "Extracting top-K insights from multi-dimensional data," in *Proc. ACM Int. Conf. Manage. Data*, May 2017, pp. 1509–1524.

[13] Y. Luo, X. Qin, N. Tang, and G. Li, "DeepEye: Towards automatic data visualization," in *Proc. IEEE 34th Int. Conf. Data Eng. (ICDE)*, Apr. 2018, pp. 101–112.

[14] F. Ojo, R. A. Rossi, J. Hoffswell, S. Guo, F. Du, S. Kim, C. Xiao, and E. Koh, "VisGNN: Personalized visualization recommendationvia graph neural networks," in *Proc. ACM Web Conf.*, Apr. 2022, pp. 2810–2818.

[15] 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.

[16] H. A. Khan and M. A. Sharaf, "Progressive diversification for column-based data exploration platforms," in *Proc. IEEE 31st Int. Conf. Data Eng.*, Apr. 2015, pp. 327–338.

[17] UCI Machine Learning Repository. *Diabetes 130 US hospitals 1999–2008 (UCI Machine Learning Repository)*. [Online]. Available: https://archive.ics.uci.edu/dataset/296/diabetes+130+us+hospitals+for+years+1999+2008

[18] T. Hicklin, *Factors Contributing to Higher Incidence of Diabetes for Black Americans*. Bethesda, MD, USA: U.S. National Institutes of Health, 2018.

[19] U.S. Departmenet of Health and Human Services Office of Minority Health. *Diabetes and African Americans*. [Online]. Available: https://minorityhealth.hhs.gov/omh/browse.aspx?lvl=4&lvlid=18#:~:text=In%202018%2C%20African%20American%20adults,complications%20than%20non%2DHispanic%20whites

[20] X. Zhang, X. Ge, P. K. Chrysanthis, and M. A. Sharaf, "ViewSeeker: An interactive view recommendation framework," *Big Data Res.*, vol. 25, Jul. 2021, Art. no. 100238.

[21] C. Wang and K. Chakrabarti, "Efficient attribute recommendation with probabilistic guarantee," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2018, pp. 2387–2396.

[22] T. Siddiqui, A. Kim, J. Lee, K. Karahalios, and G. A. Parameswaran, "Effortless data exploration with zenvisage: An expressive and interactive visual analytics system," *Proc. VLDB Endowment*, vol. 10, no. 4, pp. 457–468, 2016.

[23] F. B. Viegas, M. Wattenberg, F. van Ham, J. Kriss, and M. McKeon, "ManyEyes: A site for visualization at internet scale," *IEEE Trans. Vis. Comput. Graphics*, vol. 13, no. 6, pp. 1121–1128, Nov. 2007.

[24] T. Sellam and M. Kersten, "Fast, explainable view detection to characterize exploration queries," in *Proc. 28th Int. Conf. Sci. Stat. Database Manage.*, Jul. 2016, pp. 1–12.

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

[26] K. Zheng, H. Wang, Z. Qi, J. Li, and H. Gao, "A survey of query result diversification," *Knowl. Inf. Syst.*, vol. 51, no. 1, pp. 1–36, Apr. 2017.

[27] C. L. A. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon, "Novelty and diversity in information retrieval evaluation," in *Proc. 31st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Jul. 2008, pp. 659–666.

[28] M. Drosou and E. Pitoura, "Search result diversification," *ACM SIGMOD Rec.*, vol. 39, no. 1, pp. 41–47, Sep. 2010.

[29] D. Rafiei, K. Bharat, and A. Shukla, "Diversifying web search results," in *Proc. 19th Int. Conf. World wide web*, Apr. 2010, pp. 781–790.

[30] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. Traina, and V. J. Tsotras, "On query result diversification," in *Proc. IEEE 27th Int. Conf. Data Eng.*, Apr. 2011, pp. 1163–1174.

[31] M. Zhang and N. Hurley, "Avoiding monotony: Improving the diversity of recommendation lists," in *Proc. ACM Conf. Recommender Syst.*, Oct. 2008, pp. 123–130.

[32] Z. Hussain, H. A. Khan, and M. A. Sharaf, "Diversifying with few regrets, but too few to mention," in *Proc. 2nd Int. Workshop Explor. Search Databases Web*, May 2015, pp. 27–32.

[33] C. Yu, L. Lakshmanan, and S. Amer-Yahia, "It takes variety to make a world: Diversification in recommender systems," in *Proc. 12th Int. Conf. Extending Database Technol., Adv. Database Technol.*, Mar. 2009, pp. 368–378.

[34] Q. T. Tran and C.-Y. Chan, "How to ConQueR why-not questions," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 2010, pp. 15–26.

[35] V. Kantere, "Query similarity for approximate query answering," in *Proc. DEXA*, 2016, pp. 355–367.

[36] V. Kantere, G. Orfanoudakis, A. Kementsietsidis, and T. Sellis, "Query relaxation across heterogeneous data sources," in *Proc. 24th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2015, pp. 473–482.

[37] R. Fagin, R. Kumar, and D. Sivakumar, "Comparing top k lists," *ACM SIAM*, vol. 17, no. 1, pp. 134–160, 2003.

[38] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top- k query processing techniques in relational database systems," *ACM Comput. Surv.*, vol. 40, no. 4, pp. 1–58, Oct. 2008.

[39] Y. Hu, S. Sundara, and J. Srinivasan, "Estimating aggregates in time-constrained approximate queries in Oracle," in *Proc. 12th Int. Conf. Extending Database Technol., Adv. Database Technol.*, Mar. 2009, pp. 1104–1107.

[40] Transtats. *Transtats (US Bureau of Transportation Statistics)*. [Online]. Available: https://www.transtats.bts.gov

[41] S. Chaudhuri, "An overview of query optimization in relational systems," in *Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. Princ. Database Syst. (PODS)*, 1998, pp. 34–43.

[42] UCI Machine Learning Repository. *Heart Disease Dataset (UCI Machine Learning Repository)*. [Online]. Available: https://archive.ics.uci.edu/dataset/45/heart+disease

[43] B. Smyth and P. McClave, "Similarity vs. diversity," in *Proc. ICCBR*, 2001, pp. 347–361.

[44] L. D. Stefani, F. L. Spiegelberg, T. Kraska, and E. Upfal, "VizRec: A framework for secure data exploration via visual representation," 2018, *arXiv:1811.00602*.

[45] K. Hu, M. A. Bakker, S. Li, T. Kraska, and C. Hidalgo, "VizML: A machine learning approach to visualization recommendation," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, May 2019, pp. 1–12.

[46] V. Dibia and Ç. 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.

[47] L. Shen, E. Shen, Z. Tai, Y. Xu, and J. Wang, "Visual data analysis with task-based recommendations," 2022, *arXiv:2205.03183*.

[48] B. Mutlu, E. Veas, and C. Trattner, "VizRec: Recommending personalized visualizations," *ACM Trans. Interact. Intell. Syst.*, vol. 6, no. 4, pp. 1–39, Dec. 2016.

[49] K. Dimitriadou, O. Papaemmanouil, and Y. Diao, "AIDE: An active learning-based approach for interactive data exploration," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 2842–2856, Nov. 2016.

[50] Y. Luo, X. Qin, N. Tang, G. Li, and X. Wang, "DeepEye: Creating good data visualizations by keyword search," in *Proc. Int. Conf. Manage. Data*, May 2018, pp. 1733–1736.

[51] X. Qin, Y. Luo, N. Tang, and G. Li, "Making data visualization more efficient and effective: A survey," *VLDB J.*, vol. 29, no. 1, pp. 93–117, Jan. 2020.

[52] A. Wu, Y. Wang, X. Shu, D. Moritz, W. Cui, H. Zhang, D. Zhang, and H. Qu, "AI4 VIS: Survey on artificial intelligence approaches for data visualization," *IEEE Trans. Vis. Comput. Graphics*, vol. 28, no. 12, pp. 5049–5070, Dec. 2022.

[53] H. A. Khan, M. A. Sharaf, and A. Albarrak, "DivIDE: Efficient diversification for interactive data exploration," in *Proc. 26th Int. Conf. Sci. Stat. Database Manage.*, Jun. 2014, pp. 1–12.

[54] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong, "Diversifying search results," in *Proc. 2nd ACM Int. Conf. Web Search Data Mining*, Feb. 2009, pp. 5–14.

[55] J. Carbonell and J. Goldstein, "The use of MMR, diversity-based reranking for reordering documents and producing summaries," in *Proc. 21st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retr.*, Aug. 1998, pp. 335–336.

[56] M. M. Islam, M. Asadi, S. Amer-Yahia, and S. B. Roy, "A generic framework for efficient computation of top-k diverse results," *VLDB J.*, vol. 32, no. 4, pp. 737–761, Jul. 2023.

[57] R. Mafrur, A. M. Sharaf, and G. Zuccon, "Quality matters: Understanding the impact of incomplete data on visualization recommendation," in *Proc. DEXA*, 2020, pp. 122–138.

MOHAMED A. SHARAF received the Ph.D. degree in computer science from the University of Pittsburgh, in 2007. He is currently an Associate Professor in computer science with United Arab Emirates University (UAEU), where he joined, in 2019. Before that, he was a Senior Lecturer with The University of Queensland and a Research Fellow with the University of Toronto. His research interests include data science, with a special emphasis on large-scale big data analytics, interactive human-in-the-loop data exploration, and scalable data visualization.

RISCHAN MAFRUR received the B.Eng. degree in computer engineering from Sunan Kalijaga State Islamic University, Indonesia, and the M.Eng. degree in computer engineering from Chonnam National University, South Korea. He is currently pursuing the Ph.D. degree with the Data Science Group, The University of Queensland. He received the Indonesia Endowment Fund for Education (LPDP) Scholarship for his Ph.D. studies. His research interests include data exploration and data visualization. He is also a member of the Data Science Group, The University of Queensland.

GUIDO ZUCCON is currently an Associate Professor with the School of Information Technology and Electrical Engineering, The University of Queensland. He was an ARC DECRA Fellow (2018–2020). He leads the Information Engineering Laboratory (IELab), a research team that works in information retrieval and health data science. His research interests include information retrieval, health search, the formal models of search and search interaction, and health data science.