DiVE: Diversifiying View Recommendation for Visual Data Exploration

ABSTRACT

To support effective data exploration, there has been a growing interest in developing solutions that can automatically recommend data visualizations that reveal interesting and useful data-driven insights. In such solutions, a large number of possible data visualization views are generated and ranked according to some metric of importance (e.g., a deviation-based metric), then the top-k most important views are recommended. However, one drawback of that approach is that it often recommends similar views, leaving the data analyst with a limited amount of gained insights. To address that limitation, in this work we posit that employing diversification techniques in the process of view recommendation allows eliminating that redundancy and provides a good and concise coverage of the possible insights to be discovered. To that end, we propose a hybrid objective utility function, which captures both the importance, as well as the diversity of the insights revealed by the recommended views. While in principle, traditional diversification methods (e.g., Greedy Construction) provide plausible solutions under our proposed utility function, they suffer from a significantly high query processing cost. In particular, directly applying such methods leads to a "process-first-diversify-next" approach, in which all possible data visualization are generated first via executing a large number of aggregate queries. To address that challenge, we propose an integrated scheme called *DiVE*, which efficiently selects the top-k recommended view based on our hybrid utility function. *DiVE* leverages the properties of both the importance and diversity metrics to prune a large number of query executions without compromising the quality of recommendations. Our experimental evaluation on real datasets shows the performance gains provided by DiVE.

1 INTRODUCTION

The need for effective visual data exploration is gaining wider recognition, where automated solutions are provided to support users from professional data analysts in industry and science to data enthusiasts who lack formal training in data analytics. The goal is to enable data-driven discoveries, wherein interesting insights are unearthed from large volumes of collected data. As such, recent years have seen the introduction of many visual analytic tools such as Tableau, Qlik and Spotfire. These tools aim to provide aesthetically high-quality visualizations in terms of charts, which are essentially aggregated views of the underlying data (e.g., bar charts). For instance, the commercial Tableau visualization tool presents users with aggregate charts, with the expectation that some of those charts would reveal insights that a user finds interesting. Clearly, however, manually looking for insights in each visualization is a labor-intensive and time-consuming.

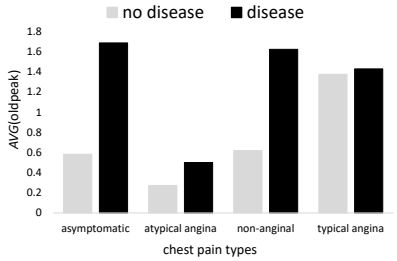


Figure 1: `AVG`(oldpeak) vs. chest pain types

Such challenge motivated multiple research efforts that focused on automatic recommendation of visualizations based on some metrics that capture the utility of a recommended visualizations (e.g., [3, 8, 11, 13–15, 18–20]). For instance, recent case studies have shown that a *deviation-based* formulation of that utility is able to provide analysts with interesting visualizations that highlight some of the particular trends of the analyzed datasets [18, 19]. Differently from Tableau's user-driven approach, in that deviation-based data-driven approach, certain views of a query result (i.e., *target view*) are recommended if they deviate significantly from those exhibited by a reference dataset (i.e., *reference view*). The intuition is that a view with high deviation is expected to reveal some important insights that are very particular to the data subset under analysis.

For instance, consider a data analyst trying to gain some insights into the *Cleveland heart disease* dataset[1]. Naturally, a first step in that exploratory analysis is to conduct some comparison between patients with heart disease and patients without heart disease. Hence, the analyst writes an SQL query that selects patients with heart disease (i.e., `disease`) as the target data subset for analysis, and the remaining patients are selected as the reference data subset (i.e., `no-disease`) .

Since the analyzed data contains different dimensions (e.g., chest pain types, sex, etc.) and different measures (oldpeak, age, etc.), it is a challenging task for the analyst to manually select the combinations of dimensions and measures that reveal interesting insights. Hence, to automatically recommend interesting bar chart visualizations, different SQL aggregate functions are applied on the views generated from all the possible pairwise combinations of dimensions and measures, then the most *important* views are presented to the analyst. For this particular example, Figure 1 shows the top-1 recommended view according to the deviation-based metric [18, 19]. The figure shows that an aggregate view (i.e., bar chart) based on *average oldpeak* (i.e., pressure of the ST segment) vs. *chest pain types* exhibits a large deviation between the target view (`disease`) and reference view (`no-disease`).

[1] http://archive.ics.uci.edu/ml/datasets/heart+Disease

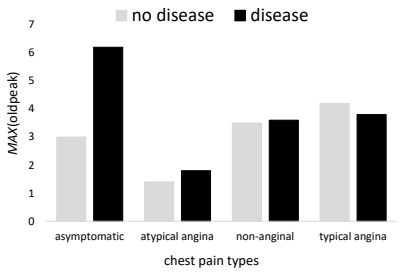


Figure 2: MAX(oldpeak) vs. chest pain types

That is, patients with heart disease often suffer more from asymptomatic and non-angina chest pains, in comparison to those without heart disease.

While recommending views based on their importance has been shown to reveal some interesting insights [3, 18, 19], such approach still suffers from a "tunnel vision", where it often recommends similar and redundant views. For instance, Figure 2 shows the second top recommended view for the analysis described above. Comparing Figures 2 and 1, it is easy to see that both visualizations are based on the same dimension (i.e., *chest pain types*) and the same measure (i.e., *oldpeak*), and the only difference between them is the aggregate function (i.e., `MAX` vs `AVG`). Despite that similarity between the two views, they are still both recommended to the analyst due to the high deviation between the target and reference views.

To address that limitation, in this work we posit that employing *diversification* techniques in the process of view recommendation allows eliminating that redundancy and provides full coverage of the possible insights to be discovered. In fact, diversity is rapidly becoming one of the fundamental features for maximizing information gain in web search and recommendation engines (e.g., [1, 12, 23, 24]). Similarly, it is highly desirable to recommend views that reveal interesting insights, while at the same time provide the analyst with a broad scope of those insights.

To that end, we propose a hybrid objective utility function, which captures both the importance, as well as the diversity of the insights revealed by the recommended views. The main goal is to select and recommend top-k views that balance the tradeoff between importance and diversity based on that hybrid objective function.

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 overall objective function (e.g., [12, 23, 24]). However, differently from assessing relevance, evaluating the importance of a view is a computationaly expensive operation, which requires the execution of rather data-intensive queries. As such, directly applying those methods leads to a "process-first-diversify-next" approach [12, 24], in which all possible data visualization are generated first via executing a large number of aggregate queries. To address that challenge and minimize the incurred query processing cost, we propose an integrated scheme called *DiVE*, 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. The main contributions of this paper are summarized as follows:

- We formulate the problem of recommending views that are both important and diverse based on a hybrid objective function, which balances the tradeoff between the *content* and the *context* of recommended view (**Section 3**).
- We propose the novel *DiVE* schemes, which employ several algorithms to select the recommended visualizations based on our hybrid ranking/objective function (**Section 4**).
- We propose novel optimization 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 4**).
- We conduct an extensive experimental evaluation on real datasets, which compare the performance of various algorithms and illustrate the benefits achieved by *DiVE* (**Sections 5 & 6**).

2 PRELIMINARIES AND RELATED WORK

Several recent research efforts have been directed to the challenging task of recommending aggregate views that reveal interesting data-driven insights (e.g., [3, 18, 19]). As in previous work, we assume a similar model, in which a visual data exploration session 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 simply defined as: Q: `SELECT * FROM` D_B `WHERE` T`;`

Ideally, the analysts would like to generate some aggregate views (e.g., bar charts or scatter plots) that unearth some valuable insights from the selected data subset D_Q. However, achieving that goal is only possible if the analyst knows exactly what to look for! That is, if they know the parameters, which specify some aggregate views that lead to those valuable insights (e.g., aggregate functions, grouping attributes, etc.). Meanwhile, such parameters only become clear in "hindsight" after spending long time exploring the underlying database. Hence, the goal of existing work, such as [3, 11, 18–20], is to *automatically* recommend such aggregate views.

To specify and recommend such views, as in previous work, 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 measure attributes, such as `COUNT, AVG, SUM, MIN and MAX`. Hence, specifying different combinations of dimension and measure attributes along with various aggregate functions, generates a set of possible views $\mathbb{V}$ over the selected dataset D_Q. For instance, a possible 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 $F_i \in \mathbb{F}$, and it can be formally defined as: V_i: `SELECT` A_i`,` F_i `(`M_i`) FROM` D_B `WHERE` T `GROUP BY` A_i;

Clearly, an analyst would be interested in those views that reveal interesting insights. However, manually looking

for insights in each view $V_i \in \mathbb{V}$ is a labor-intensive and time-consuming process. For instance, consider again our example in the previous section. In that example, let D_B be the Cleveland Heart Disease table (i.e., `tb_heart_disease`) and the analyst is selecting the subset of patients with heart disease (i.e., D_Q `= disease` subset). Hence, the number of views to explore is equal to: $|\mathbb{V}| = |\mathbb{A}| \times |\mathbb{M}| \times |\mathbb{F}|$, where $|\mathbb{F}|$ is the number of SQL aggregate functions, and $|\mathbb{A}|$ and $|\mathbb{M}|$ are the number of attribute and measures in `tb_heart_disease`, respectively. For that medium-dimensionality dataset, that value of $|\mathbb{V}|$ goes up to 180 views, which is clearly unfeasible for manual exploration. 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., [3, 8, 11, 13–15, 18–20]).

Those approaches can be broadly classified as *user-driven* or *data-driven*. User-driven solutions focus on recommending visualizations that facilitate a particular user intent or task. For example, VizDeck [11] utilizes user feedback as a basis for view recommendation, whereas Profiler [8] detects anomalies and recommends visualizations based on mutual information metric. Similarly, Rank-by-Feature Framework [15] enables users to select their criterion for ranking histograms and scatter-plots.

Meanwhile, *data-driven* focus on enabling the discovery of interesting insights from large volumes of data without requiring much prior knowledge of the explored data. Towards that end, data-driven metrics are employed to capture the *interestingness* or *importance* of a recommended visualization. Recent case studies have shown that a *deviation-based* metric is effective in providing analysts with *important* visualizations that highlight some of the particular trends of the analyzed datasets [3, 4, 18, 19].

In particular, the deviation-based metric measures the distance between $V_i(D_Q)$ and $V_i(D_R)$. That is, it measures the deviation between the aggregate view V_i generated from the subset data D_Q vs. that generated from a reference dataset D_R, where $V_i(D_Q)$ is denoted as *target* view, whereas $V_i(D_R)$ is denoted as *reference* view. That reference dataset could be the whole database (i.e., $D_R = D_B$) or a selected subset of the database (e.g., Sec. 1). 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. In case, $D_R = D_B$, then the patterns extracted from D_Q are fundamentally different from the generals ones manifested in the entire database D_B.

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 the possible insights. As illustrated in the previous section, Figures 1 and 2 show two recommended views that basically reveal the same insight. To address that limitation, in this work we posit that employing *diversification* techniques in the process of view recommendation allows eliminating that redundancy and provides a good and concise coverage of the possible insights to be discovered. In the next section, we discuss in details the formulation of both importance and diversity, and their impact on the view recommendation process.

3 DIVERSIFYING RECOMMENDED VIEWS

Towards formulating our hybrid objective for view recommendation, in this section we describe the *content-based* deviation metric for assessing the importance of each view (Sec. 3.1), together with our *context-based* measure of the (dis)similarity between different views (Sec. 3.2). Those two metrics provide the foundations for our hybrid objective that aims to balance the tradeoff between importance and diversity in view recommendation (Sec. 3.3).

3.1 Content-Driven Importance

As briefly described in the previous section, in this work we adopt a deviation-based metric to quantify the importance of an aggregate view [18, 19]. 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 $V_i(D_R)$).

Clearly, the deviation between a target and a reference view is a *data-driven* metric. That is, it measures the deviation between the aggregate *result* of $V_i(D_Q)$ and that of $V_i(D_R)$. Consequently, and from a visualization point of view, that deviation is a *content-based* metric that captures the difference between the content of the visualization generated by $V_i(D_Q)$ vs. the visual content generated from $V_i(D_R)$. In the next, we formally describe the standard computation of that data-driven content-based metric, whereas the discussion of its counterpart context-driven metric is deferred to the next section.

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 $P[V_i(D_R)]$. In particular, consider an aggregate view $V_i =< A_i, M_i, F_i >$. The result of that view can be represented as the set of tuples: $<(a_j, g_j), (a_j, g_j), ..., (a_t, g_t)>$, where t is the number of distinct values (i.e., groups) in attribute A_i, a_j is the j-th group in attribute A_i, and g_j is the aggregated value $F_i(M_i)$ for the group a_j [3, 18]. Hence, V is normalized by the sum of aggregate values $G = \sum_{j=1}^{t} g_j$, resulting in the probability distribution $P[V_i] =< \frac{g_1}{G}, \frac{g_2}{G}, ..., \frac{g_t}{G} >$.

Finally, 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 3), and is simply defined as:

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

where $I(V_i)$ is the importance score of V_i and *dist* is a distance function. Similar to existing work (e.g., [18, 19]), we adopt a Euclidian distance function, but other distance measures are

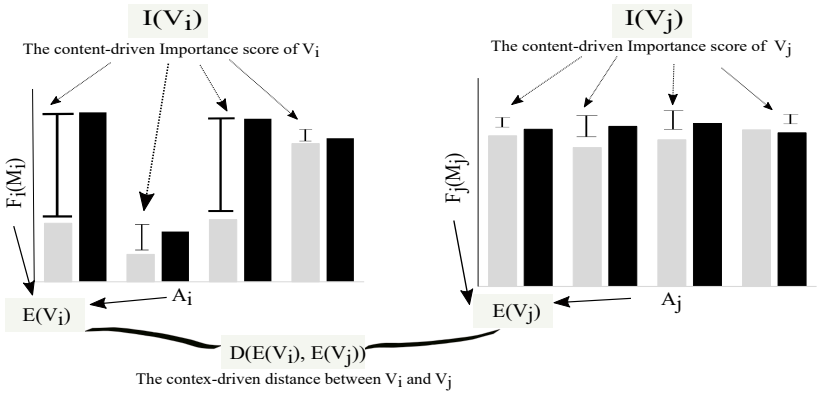


Figure 3: Content vs. Context of views.

also applicable (e.g., Earth Mover's distance, K-L divergence, etc.).

In current approaches for view recommendation, the importance value $I(V_i)$ of each possible view V_i is computed, and the k views with the highest deviation are recommended (i.e., *top-k*) (e.g., [18]). However, in this work, our goal is to ensure that recommended views provide a good coverage of possible insights, which is achieved by considering the context of the recommended views, which is described next.

3.2 Context-Driven Similarity

As mentioned above, recommending top-k views based only on their data content (i.e., content-driven deviation) often leads to a set of similar views. In order to provide full coverage of all possible interesting insights, in this work, we posit that achieving *diversity* within the set of recommended views is an essential quality measure. Diversity has been well known and widely used in recommendation systems for maximizing information gain and minimizing redundancy (e.g., [12, 21, 23, 24]). At a high level, diversity essentially measures how different (i.e., diverse) are the individual data objects within a set.

Before discussing the details of diversity computation in Sec. 3.3, it is important to notice that central to that computation is some notion of distance measure between data objects. Existing work provides multiple metrics for measuring that distance between traditional data objects, such as web documents (e.g., [1, 12, 24]), database tuples (e,g,, [17]), etc. However, our work in this paper is the first to consider diversity in the context of aggregate data visualizations. As such, a metric is needed to capture and quantify the (dis)similarity between the distinct features of different visualizations. Towards this, we re-emphasize that each visualization is merely a data view generated by an SQL aggregate query. Thus, such metric naturally lends itself to considering the query underlying each view. That is, the query that has been executed to create the visualization. In turn, the distance between two visualizations is measured based on the distance between their underlying queries. Hence, in addition to the data-driven content-based deviation described above, here we also introduce a query-driven *context-based* deviation metric.

Towards measuring the context-based deviation, we extend on existing work in the area of query recommendation and refinement (e.g., [9, 10, 17]). In that work, the distance between two range queries q_1 and q_2 is mapped to that of measuring the edit distance needed to transform q_1 into q_2, where the set of allowed transformation are: add, delete, or modify a predicate. In the context of our work, however, views are generated from aggregate queries without range predicates. In particular, a view is fully defined in terms of a combination of attribute, measure and an aggregate function. Hence, 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. 3.1), 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 (e.g., [9, 10, 17]), 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: $J(V_i, V_j) = \dfrac{|E(V_i) \cap E(V_j)|}{|E(V_i) \cup E(V_j)|}$

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, let w_i be the weight assigned to i^{th} element of set $E(V_i)$, where $\sum_{i=1}^{3} w_i = 1$. Then, the

similarity between views V_i and V_j is measured as: $J(V_i, V_j) = \frac{\sum_{i \in V_i \cap V_j} w_i}{\sum_{i \in V_i \cup V_j} w_i}$

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

$$D(V_i, V_j) = 1 - J(V_i, V_j) \quad (2)$$

Figure 3 illustrates and summarizes our proposed metrics. Particularly, it shows two views V_i and V_j, for which the content-driven importance values $I(V_i)$ and $I(V_j)$ are computed based on the probability distribution of the data in each view. Moreover, it also shows the context-driven distance between those two views, which is computed as $D(V_i, V_j)$.

3.3 Problem Definition

In this section, we formally define our problem for recommending diversified interesting aggregate views. Towards this, we first define the metrics to measure the performance of our proposed visualization recommendation system in terms of: 1) the quality of recommended visualizations, and 2) the processing cost incurred in computing those visualizations.

3.3.1 Hybrid Objective Function. Our hybrid objective function is designed to consider both the importance and diversity of the recommended views. Particularly, it integrates two components: 1) the total importance score of set S and 2) the diversity score of S.

The importance score of the a S is calculated as the average value of the importance measure of each view in S, as given in Eq.1. 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, which is achieved when for each group a_i, the corresponding value $\frac{g_i}{G}$ in $P[V_i(D_R)]$ or $P[V_i(D_Q)]$ is zero. Thus, $I_u = \sqrt{2}$, and is used to normalize the average importance score for set S.

In order to measure the diversity of a set of objects, several diversity functions have been employed in the literature [1, 21]. Among those, previous research has mostly focused on measuring diversity based on either the average or the minimum of the pairwise distances between the elements of a set [22]. In this work, we focus on the first of those variants (i.e., average), as it maximizes the coverage of S. Hence, given a distance metric $D(V_i, V_j)$, as given in equation 2, the diversity of a set S can be simply measured as follows:

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

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

Next, we define our proposed hybrid objective function that captures both the importance and diversity of the set of recommended views S. Specifically, for a set of views $S \subseteq V$, 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:

$$F(S) = (1 - \lambda) \times I(S) + \lambda \times f(S, D) \quad (3)$$

where $0 \leq \lambda \geq 1$ is employed to control the preference given to each of the importance and diversity components. For instance, a higher value of λ results in a set of more diverse views, whereas a lower value of λ generates a set of the most important views, which is likely to exhibit redundancy in the recommended views.

Given the hybrid objective function, our goal is to find an optimum set of views S^* that maximizes the objective function $F(S)$, which is 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 $F(S)$ is maximized.*

Given the definition above, the quality of the recommended set of views is measured in terms of the value of the hybrid objective function $F(S)$.

3.3.2 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 [3, 18, 19]. Moreover, integrating diversification to 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 simple 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 overall utility function formulated in Eq.3. For instance, in the context of recommending web search, such methods are designed to recommend a set of diversified objects (e.g., web documents) that are relevant to the user needs. However, in that setting, the relevance of an object is either given or simply computed. To the contrary, in our setting for view recommendation, the importance of a view is a computational expensive operation, which requires the execution of a target and reference view. As such, directly applying those methods leads to a "process-first-diversify-next" approach, in which all possible data visualization are generated first via executing a large number of aggregate queries. To address that challenge and minimize the incurred query processing cost, in the next section we propose an integrated scheme called *DiVE*, leverages the properties of both the importance and diversity to prune

a large number of a large number of low-utility views, without compromising the quality of recommendations.

4 THE DIVE SCHEMES

In this section, we present our DiVE schemes for recommending diversified top views, which is captured by Eq. 3, while at the same time minimizing the incurred processing costs. Towards this, we first simply expand on the well-known *Greedy* heuristic for data diversification and propose our DiVE-Greedy scheme (Sec. 4.2). In Sec. 4.3, we propose novel pruning strategy that allows reducing the overall DiVE-Greedy processing cost. To overcome the *constructive* nature of DiVE-Greedy and allow further reductions in processing cost, we propose our *interchange local-search* method DiVE-Swap in sections 4.4 and 4.5. Finally, we introduce our *adaptive* pruning method, which is based on *non-parametric predictive intervals* and strikes a fine balance between the efficiency and effectiveness in view recommendation.

4.1 Baseline Solutions

As baseline solutions to compare the performance of our proposed DiVE schemes, we simply incorporate methods from existing work that optimize either for importance or diversity. In terms of diversity, we employ the classical *Greedy Construction* algorithm [16], which has been shown to maximize diversity within reasonable bounds compared to the optimal solution [21, 23]. 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 [18, 19]. 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). Clearly, those two methods are "oblivious" to our hybrid objective function (i.e., Eq.3). In particular, as shown in our experimental evaluation (Sec. 6), each of those two methods performs well under extreme settings of our hybrid function. As expected, Greedy-Diversity provides its best performance when $\lambda = 1.0$ (i.e., all preference is given to diversity), whereas Linear-Importance is the winner when $\lambda = 0.0$ (i.e., all preference is given to importance). In the following, we present our DiVE schemes which are able to provide the best performance (i.e., maximize the overall hybrid objective), irrespective of the value of λ, while minimizing the processing time.

4.2 The DiVE-Greedy Scheme

In this section, we discuss our first DiVE scheme (*DiVE-Greedy*), which simply extends the basic Greedy Construction algorithm to work under our hybrid objective function (i.e., Eq. 3). Such extension is straightforward and is described in Algorithm 1. Similar to the classical Greedy Construction, DiVE-Greedy initializes the set S with the two most distant views, where the distance between any two views is calculated using our context-based function, as given in Eq.2. Then,

Algorithm 1: *DiVE* Greedy

Input: Set of views V and result set size k
Output: Result set $S \geq V$, |S| = k

```
S ← [V_i, V_j] get two most distant views;
X ← [V\S];
i ← len(S);
while |S| < k do
    v_i ← argmax (1 − λ) × I(v_i) + λ × setDist(v_i, S);
    S.add(v_i);
    X.remove(v_i);
end
return S
```

DiVE-Greedy iteratively selects new views to be added to S. Particularly, in each iteration a view is selected from the set of remaining views X and is added to S. To make that selection, DiVE-Greedy assigns a score to each view in X, which is based on the hybrid objective function $F(S)$, as defined in equation 3. Specifically, the utility score assigned to a view $V_i \in X$ is computed as:

$$U(V_i) = (1-\lambda) \times I(V_i) + \lambda \times setDist(V_i, S) \qquad (4)$$

where

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

Thus, in each iteration, the view with highest utility score is selected and added to S, until $|S| = k$, as shown in Algorithm 1.

DiVE-Greedy Cost: We note that the only difference between our DiVE-Greedy scheme and our baseline Greedy-Diversity (i.e., the classical Greedy Construction algorithm) is in the utility score assigned to each view (i.e., $U(V_i)$ in Eq.4). In fact, in the special case where $\lambda = 1.0$, Eq. 4 boils down to $U(V_i) = setDist(V_i, S)$, which is the same score used by Greedy-Diversity for maximizing diversification. However, that simple change in the utility score leads to executing the query underlying each view V_i in order to compute the $(1-\lambda) \times I(V_i)$ component of its score. Hence, the overall cost of DiVE-Greedy is $C_T = C_Q + C_D$, as opposed to the cost of Greedy-Diversity, which is only $C_T = C_D$, 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. 3.

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 is $O(kn)$, where k is the number of recommended views. Hence, as presented, our extended DiVE-Greedy is an instance of what has been described earlier as a "process-first-diversity-next" approach [12, 24], in which diversification only takes place after all possible views are executed to calculate their importance. In the next section, we describe our work to overcome the limitations of that approach.

4.3 DiVE-Greedy with Pruning

As mentioned in the previous section, the cost of the DiVE-Greedy algorithm is dominated by the query processing cost C_Q and is proportional to the number of possible views. Although hundreds of views are generated for a given subset of data D_Q, only a small fraction of those views are actually of interest and are candidates to be included in the top-k set. As such, a significant fraction of the query processing cost is incurred in evaluating low-utility views. This observation motivated us to propose a pruning technique to minimize the search space of views and narrow it down to the most promising ones, as follows.

Our proposed pruning technique is based on the observation that the utility score of each view $U(V_i)$ is a weighted sum of two measures; 1) the importance score of the view (i.e., $I(V_i)$), and 2) the distance of the view from S (i.e., $setDist\,(V_i, S)$). We note that the computation of $setDist\,(V_i, S)$ is a CPU-bound requires fast operation. To the contrary, computing the importance score of a view $I(V_i)$ is an expensive operation that requires executing two queries to generate the target and reference data for V_i. Thus, we employ a *max-min* pruning technique, that leverages the diversity score to bound the maximum utility score achieved by each view V_i, and in turn allows for pruning low-utility views without incurring the high cost for evaluating their importance.

Particularly, max-min utilizes the maximum and minimum bound on the importance score $I(V_i)$. Clearly, the minimum utility score is $I_0 = 0$, whereas the maximum utility value I_u a view can score is computed as $I_u = \sqrt{2}$ (Sec. 3.3). Using those bounds, in each iteration the maximum $maxU(V_i)$ and minimum utility $minU(V_i)$ score for each view $V_i \in X$ is calculated as:

$$maxU\,(V_i) = (1-\lambda) \times I_u\,(V_i) + \lambda \times setDist\,(V_i, S)$$

$$minU\,(V_i) = (1-\lambda) \times I_0\,(V_i) + \lambda \times setDist\,(V_i, S)$$

Accordingly, the maximum value of $minU$ is recorded (i.e., maximum of mimumum). Then, if $maxU$ of any candidate view is less than the maximum of $minU$, then that view is pruned.

4.4 The DiVE-Swap Scheme

Naturally, our DiVE-Greedy scheme described above will achieve a high pruning power for high values of λ (i.e., more emphasize is given to the diversity score $setDist\,(V_i, S)$). However, that pruning power diminishes for low values of λ (i.e., more emphasize is given to the importance score $I(V_i)$). That discrepancy in performance happens because of two reasons: (1) the constructive and iterative nature of the greedy algorithm, and (2) utilizing the upper bound on importance I_u when computing $maxU(V_i)$. To explain those reasons, recall that the amount of pruning is shaped by the maximum value of $minU(V_i)$. Now, consider an extreme case, in which $\lambda = 0.0$. In that case, all values of $minU(V_i)$ will also be equal to zero, and no pruning will take place. Similarly, for low values of $\lambda > 0$, the maximum value of $minU(V_i)$ will still be low, leading to limited pruning. That pruning is further reduced when combined with a high value of I_u as it leads to all views acquiring high $maxU(V_i)$ and cannot be pruned. To address those limitations, we propose an adaptive scheme for setting I_u, which is described in Sec. 4.6, and we orthogonally introduce our *Swap*-based DiVE scheme, which is described next.

Algorithm 2: *DiVE* Swap

Input: Set of views V and result set size k
Output: Result set $S \geq V$, |S| = k

```
1  S ← Result set of importance or diversity maximized;
2  X ← [V\S];
3  while improve = True do
4      for i in set X do
5          S' ← S;
6          for j in set S do
7              if F(S') < F(S\S[j] ∪ X[i]) then
8                  S' ← S\j ∪ X[i] ;
9              end
10         end
11         if F(S') > F(S) then
12             S ← S'
13         end
14     end
15 end
16 return S
```

The DiVE-Greedy algorithm presented in the previous section is of the constructive type. That is, it starts with an empty set of views and incrementally constructs it by adding one view at a time. To the contrary, our DiVE-Swap presented in this section falls under the *local search* type of algorithms. In general, 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 [2, 21], and in this paper, we further expand it to our DiVE scheme.

The basic idea underlying DiVE-Swap 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 $F(S)$. One of the main design criteria in local search algorithms is the choice of the initial solution. In DiVE-Swap, we consider two natural variants: 1) *DiVE-iSwap*, and 2) *DiVE-dSwap*. In DiVE-iSwap, S is initialized with the k views that maximize importance (i.e., all preference is given to importance) and can be easily obtained using our baseline Linear-Importance (Sec. 4.1). Alternatively, in DiVE-dSwap, S is initialized with the k views that maximize diversity (i.e., all preference is given to diversity), which also can be easily obtained using our baseline Greedy-Diversity (Sec. 4.1).

Apart from the initialization approach, both variants work similarly. Particularly, in each iteration, each unselected view $X_i \in X$ is interchanged with all the selected views in $S_j \in S$ (Algorithm 2 line 9). That is, the overall hybrid objective

is computed for $F(S \backslash S_j \cup X_i)$. Then the one interchange that leads to the highest new value for F is applied and S is updated accordingly (Algorithm 2 line 10). Such iterations are repeated until no more views can be swapped between X and S, which is reached when no further improvement is achieved in the value of F.

In comparison to DiVE-Greedy, DiVE-Swap incurs the same query processing cost C_Q. Furthermore, it incurs even higher C_D cost for computing diversity, which can reach up to $O\left(kn^2\right)$ in the worst case. However, as described in the next section, DiVE-Swap offers a valuable opportunity for maximizing the number of pruned views, and in turn reducing the query processing cost C_Q, which is the dominant factor in determining the efficiency.

4.5 Optimized DiVE-dSwap

While both DiVE-iSwap and DiVE-dSwap described above incur the same cost, they offer substantially different performance when combined with pruning techniques. Particularly, consider DiVE-iSwap, which is initialized with the k views that maximize importance. To select those views, all possible views have to be generated first, which in turn requires processing all their corresponding queries. Hence, DiVE-iSwap simply eliminates all opportunities for pruning as all views are executed in the initialization phase. To the contrary, DiVE-dSwap is initialized with the k views that maximize diversity. To select that initial set, no query execution is needed and the processing is limited to computing the context-based deviation distances, which is significantly low compared to query processing. Hence, DiVE-dSwap allows integrating effective pruning techniques, as described next.

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 the overall hybrid objective $F(S)$. That is, $F(S \backslash S_j \cup X_i) > F(S)$. Hence, the task boils down to finding that pair of views $< X_i, S_j >$ that provide the maximum improvement in $F(S)$ once interchanged. Without pruning, that requires iterating through S and X simultaneously and computing F for each pair, which requires processing and generating each view in X. To avoid such expensive processing and enable pruning, the following steps are taken.

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. 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 is clearly similar to top-k query processing, for which numerous optimization techniques are proposed (e.g., [5, 7]). Particularly, to find the *top-1* view, each view X_i is initially assigned an importance equal to the upper bound I_u. In turn, the upper bound of $F(S)$ achieved by X_i is computed as: $maxF(S \backslash S_j \cup X_i)$, which is based on the actual diversity achieved by the swap, and the upper bound on importance. As such, $maxF(S \backslash S_j \cup X_i)$ is compared against $F(S)$, leading to one of the following two cases: If $maxF(S \backslash S_j \cup X_i) > F(S)$, then the swap $< X_i, S_j >$ can "potentially" improve $F(S)$. 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:

$$maxF(S \backslash S_j \cup X_i) < F(S)$$

Simply put, if the upper bound $maxF$ achieved by that swap is still less than the current $F(S)$, then the actual $F(S \backslash S_j \cup X_i)$ is guaranteed to be less than $F(S)$ and the pair $< X_i, S_j >$ can be safely ignored (i.e, pruned). More importantly, since the list of pairs is sorted by diversity, 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.

DiVE-dSwap vs. DiVE-Greedy: At this point, it is especially important to examine and contrast the pruning power achieved by each of DiVE-Greedy and DiVE-dSwap. For DiVE-Greedy, recall that pruning is attained for those views where: $maxU(V_i) < maxminU$. However, since DiVE-Greedy is a constructive algorithm, the set S is incrementally constructed iteration by iteration until $|S| = k$. Hence, in the first iterations S has a very small number of selected views with a minimum of $|S| = 2$. Naturally, when S is a small set, then most of the remaining unselected views in X are expected to exhibit high diversity, since the majority of them will be very dissimilar from the small set of views in S. Accordingly, for most of the views in X, the value $maxU(V_i)$ will be relatively high, due to achieving high score on diversity. Hence, most views will fail to satisfy the pruning condition and are consequently executed incurring high query processing cost.

To the contrary, DiVE-dSwap is initiated with a set S of k diverse views. Hence, it will initially have a reasonably high $F(S)$. Moreover, many views in X will be more or less "close" to some view in S. Hence, the swaps that involve those views will score low on diversity, and in turn low $maxF$. Since pruning happens when $maxF(S \backslash S_j \cup X_i) < F(S)$, the combination of those two factors above (i.e., high F and low $maxF$) allows for many views satisfying the pruning condition, which improves the pruning power of DiVE-dSwap over DiVE-Greedy. That pruning power can be further improved by relaxing the assumption about maximum importance I_u, as described next.

4.6 Adaptive Pruning with Predictive Intervals

In general, both the pruning schemes provided by DiVE-Greedy and DiVE-dSwap rely on the fundamental idea of evaluating the upper bound of the benefit provided by a view V_i towards the objective F. 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, both schemes compute the actual diversity offered by V_i and instead of computing its actual importance, it is substituted with the maximum attainable importance

score I_u. Naturally, overestimating $I(V_i)$ leads to overestimating its benefit and consequently limited pruning power is achieved. Meanwhile, for most datasets, I_u is in fact an overestimation of $I(V_i)$. Hence, our goal in this section to provide a tighter bound on $I(V_i)$, which allows for maximum pruning while maintaining the quality of the solution.

Recall that I_u is achieved when for each group a_i, the corresponding value $\frac{g_i}{G}$ 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 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 [6]. That challenge is further emphasized when the values exhibited by the population are skewed and do not follow a typical normal distribution, which is typically the case for the importance value of views.

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 [6]. To apply that 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 $m = 19$, results in prediction interval PI = 90%. That is, 90% of the time, the importance value of an unseen view V_i will be less than the maximum importance seen so far. Clearly, the higher the PI value, the higher the accuracy of $\bar{I}_{au}$ in estimating I_{au}, but also requires executing more views. In this work, we find that a value of $PI = 97\%$ is able to balance the tradeoff between minimizing the number of executed queries and maximizing the objective value F, as illustrated in the next sections.

5 EVALUATION TESTBED

In this section, we present our settings for evaluating the performance of our proposed *DiVE* scheme in terms of both efficiency and effectiveness. Table 1 summarizes the different parameters used in our experimental evaluation while the default values are presented in bold.

Table 1: Parameters testbed in the experiments

Parameter	Range (**default**)
datasets	**Heart disease**, Flights
sample queries	**10**
diversity weight ratio	**3**(A) **: 2**(M) **: 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
prediction interval %	80 , 85, 90, 95, **97**, 98

We have conducted our experiments over following real datasets: 1) Heart Disease Dataset [2]: This dataset is comprised of 9 dimensional attributes and 5 measure attributes, resulting in a total of 180 possible views, and 2) Airline (Flights) Dataset [3]: This dataset is comprised of 7 dimensional attributes and 4 measure attributes for a total of 112 views. While its dimensionality is lower than 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.

We evaluate the performance of following schemes: 1) Linear-Importance and Greedy-Diversity, which are our baseline methods (Sec. 4.1), 2) DiVE-Greedy (Sec. 4.2), 3) DiVE-iSwap and DiVE-dSwap (Sec. 4.4), which employ an interchange algorithm initialized with the most important and most diverse views, respectively, 4) DiVE-Greedy-Static and DiVE-dSwap-Static (Sec. 4.3 and 4.5), which use static pruning techniques based on the theoretical I_u (Sec. 3.3), and 5) DiVE-Greedy-Adaptive and DiVE-dSwap-Adaptive, which use adaptive pruning based on predictive intervals to estimate $\bar{I}_{au}$ (Sec. 4.6).

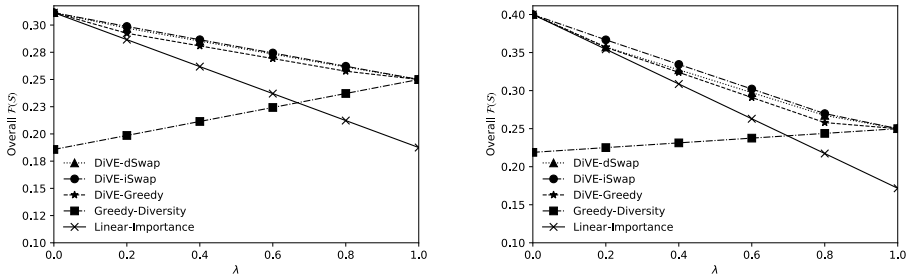
For each experiment a query workload of ten random queries is submitted to select ten different subsets of data D_Q. The performance measures are averaged over views recommended for each of those ten subsets.

6 EXPERIMENTAL EVALUATION

The impact of λ on the objective function The value of λ balances the trade off between importance and diversity score. Figure 4 shows how the performance of each scheme in terms of $F(S)$ is effected as the value of λ varies from 0 to 1. It is clearly seen in Figure 4, that for the lower values of λ the highest objective function value is achieved by Linear-Importance method. To the contrary, the Greedy-Diversity method achieves highest values of $F(S)$ as the λ approaches 1. Hence, there is a crossover between Linear-Importance and Greedy-Diversity. However, our proposed schemes based on the hybrid objective function have stable performance for all values of λ and are able to achieve $F(S)$ values higher than those achieved by Linear-Importance and Greedy-Diversity.

Execution time evaluation In this experiment we measure the cost of *DiVE* in terms of execution time. Figure 6 plots the execution time for Flights dataset with $k = 5$ and $\lambda = 0.5$. The total execution time is split into the query execution

[2] http://archive.ics.uci.edu/ml/datasets/heart+Disease
[3] http://stat-computing.org/dataexpo/2009/the-data.html



(a) Heart disease dataset **(b) Flights dataset**

Figure 4: Impact of λ on $F(S)$, k = 5

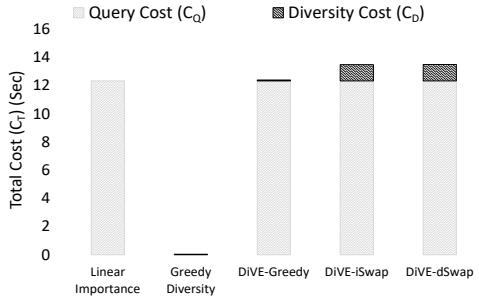


Figure 5: Cost Analysis of DiVE, k =5, $\lambda = 0.5$

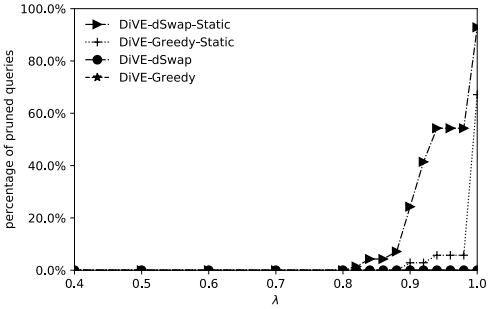


Figure 6: Impact of Static pruning

time C_Q and the diversification cost C_D. It is clear from Figure 6 that the total execution time C_T is dominated by the cost of generating the views C_Q. Hence, the minimum cost is incurred by the Greedy-Diversity method which only computes diversity. For all other methods, the C_Q component of the execution time is same as all views are generated only once. However, the cost of diversification C_D is slightly higher for DiVE-iSwap and DiVE-dSwap as compared to the DiVE-Greedy due the higher number of iterations.

Impact of static Pruning In this experiment we present the performance of our proposed pruning techniques in terms of the number of pruned queries required to generate the views. The higher number of pruned queries result in the higher cost savings in the total query execution time. Figure 7 shows the performance of our static pruning technique using the maximum value of importance score I_u. In this and next experiments, DiVE-iSwap is not evaluated as it executes all

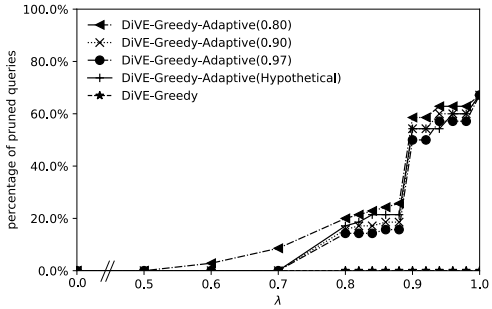


Figure 7: DiVE-Greedy with Adaptive Pruning

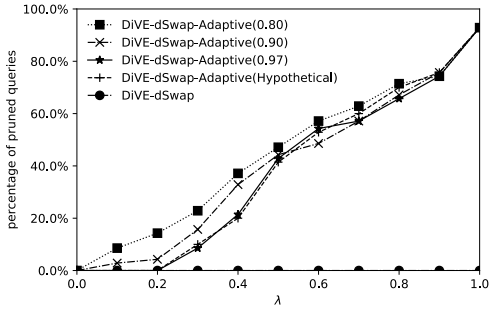


Figure 8: DiVE-dSwap with Adaptive Pruning

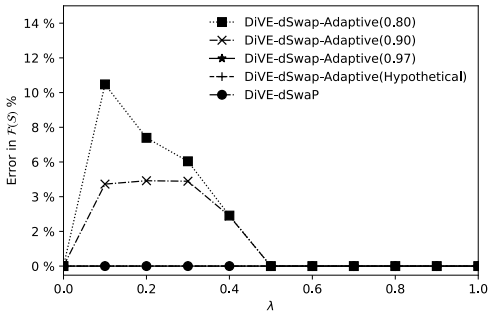


Figure 9: Impact of Adaptive pruning on $F(S)$

the view queries for the initial set selection and any pruning afterwards is not possible. Moreover, due to space limit, we use only the heart disease dataset in the next experiments.

For both schemes DiVE-Greedy-Static and DiVE-dSwap-Static, since the I_u value can be far from the actual importance scores of individual views, the percentage of pruned queries is 0 for lower values of λ. Only for λ close to 0.9 some queries get pruned. DiVE-dSwap-Static is able to prune higher number of queries than DiVE-Greedy-Static. At $\lambda = 1$, DiVE-dSwap-Static prunes 80% queries while DiVE-Greedy-Static prunes almost 65% queries. This is because for $\lambda = 1$ the weight of importance score is 0 in the hybrid objective function and the overall value of $F(S)$ is determined by the diversity score only.

Impact of Adaptive Pruning In this experiment we analyze the performance of adaptive pruning technique under different values of λ and prediction interval PI.

DiVE-Greedy-Adaptive: As shown in Figure 8, DiVE-Greedy-Adaptive is able to prune queries even for lower values of λ. The number of queries pruned increase significantly for higher values of λ. In comparison to the static pruning as shown in Figure 7, with adaptive pruning DiVE-Greedy is able to prune almost 20% more queries at $\lambda = 0.8$. The best performance is shown by DiVE-Greedy-Adaptive(0.80) as it executes a smaller number of sample queries.

DiVE-dSwap-Adaptive: Figure 9 shows the performance of DiVE-dSwap-Adaptive with different values of PI. In comparison to DivE-Greedy-Adaptive, the number of pruned queries by DivE-dSwap-Adaptive are much higher for all values of λ. The interesting observation is the fact that DiVE-dSwap-Adative is able to prune 15% queries for λ values as low as 0.2. For higher values of λ the percentage of pruned queries is between 60% and 90%. Similar to DiVE-Greedy-Adaptive, highest number of queries are pruned for $PI = 0.8$. The performance declines slightly as we increase PI to 0.97. This is because when the sample size is large, many queries are already executed and the margin of cost savings by pruning the remaining queries is small.

Further we evaluate the effectiveness of methods with adaptive pruning in terms of the $F(S)$ values. Figure 10 shows the loss in $F(S)$ in comparison to the $F(S)$ values achieved by Hypothetical methods. The loss for both DiVE-Greedy-Adaptive and DiVE-dSwap-Adaptive is 0% for $PI = 0.97$, With a larger sample size the accuracy of approximated importance score is higher. For a smaller sample size of $PI = 0.80$ there is 0% loss while $\lambda = 0$ because at the moment there are no pruned queries. However, there is a maximum loss of 10% at $\lambda = 0.1$. The loss in $F(S)$ values decrease as λ increases as the impact of importance score becomes smaller in the hybrid objective function. Meanwhile, starting $\lambda \geq 0.5$ the loss is 0%.

7 CONCLUSIONS

We proposed an effective visualization recommendation scheme that integrates both importance and diversity in the recommended views. Our proposed scheme, called *DiVE*, combines importance and diversity into a *hybrid utility function* to provide full coverage of the possible insights to be discovered. In addition to employing a hybrid utility function for effective view recommendation, *DiVE* also leverages the properties of both the importance and diversity metrics to prune a large number of query executions without compromising the quality of recommendations.

REFERENCES

[1] Charles L.A Clarke and et al. 2008. Novelty and diversity in information retrieval evaluation. *SIGIR*.
[2] M Drosou and et al. 2010. Search Result Diversification. *SIGMOD Record* 39, 1.
[3] H Ehsan et al. 2016. MuVE: Efficient Multi-Objective View Recommendation for Visual Data Exploration. *ICDE*.
[4] H. Ehsan et al. 2018. Efficient Recommendation of Aggregate Data Visualizations. *TKDE* 30, 2 (Feb 2018), 263–277.
[5] Ronald Fagin et al. 2003. Comparing Top K Lists. In *ACM-SIAM (SODA '03)*.
[6] Ying Hu and et al. 2009. Estimating Aggregates in Time-constrained Approximate Queries in Oracle. In *EDBT*. ACM, 1104–1107.
[7] Ihab F. Ilyas and et al. 2008. A Survey of Top-k Query Processing Techniques in Relational Database Systems. *ACM Comput. Surv.* 40, 4 (2008).
[8] Sean Kandel and et al. 2012. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *AFI*.
[9] V. Kantere. 2016. Query Similarity for Approximate Query Answering. In *DEXA*.
[10] Verena Kantere et al. 2015. Query Relaxation across Heterogeneous Data Sources. In *CIKM*.
[11] Alicia Key et al. 2012. VizDeck: self-organizing dashboards for visual analytics. *SIGMOD Conference*.
[12] Davood Rafiei, Krishna Bharat, and Anand Shukla. 2010. Diversifying web search results. *WWW*.
[13] Thibault Sellam et al. 2016. Ziggy: Characterizing Query Results for Data Explorers. *PVLDB* 9, 13 (2016), 1473–1476.
[14] Thibault Sellam and Martin L. Kersten. 2016. Fast, Explainable View Detection to Characterize Exploration Queries. In *SSDBM*.
[15] Jinwook Seo and Ben Shneiderman. 2006. Knowledge Discovery in High-Dimensional Data: Case Studies and a User Survey for the Rank-by-Feature Framework. *TVGC* 12, 3 (2006), 311–322.
[16] Barry Smyth and Paul Mcclave. 2001. Similarity vs . Diversity.
[17] Quoc Trung Tran and Chee-Yong Chan. 2010. How to ConQueR why-not questions. In *SIGMOD*.
[18] M Vartak et al. 2015. SEEDB : Efficient Data-Driven Visualization Recommendations to Support Visual Analytics. *VLDB* 8.
[19] Manasi Vartak and Samuel Madden. 2014. S EE DB : Automatically Generating Query Visualizations. *VLDB*.
[20] Fernanda B. Viegas and et al. 2007. Many Eyes: A site for visualization at internet scale. *TVGC* (2007), 1121–1128.
[21] Marcos R. et al. Vieira. 2011. On query result diversification. *ICDE*.
[22] Eugene Wu, Leilani Battle, and Samuel R. Madden. 2014. The case for data visualization management systems. *VLDB Endowment*.
[23] Cong Yu and et al. 2009. It Takes Variety to Make a World: Diversification in Recommender Systems. In *EDBT*.
[24] Mi Zhang and Neil Hurley. 2008. Avoiding Monotony: Improving the Diversity of Recommendation Lists. *Recommender Systems*.