Noname manuscript No.
(will be inserted by the editor)

Constrained Recommendations for Query Visualizations

Ibrahim A. Ibrahim · Abdullah Albarak · Xue Li

Received: date / Accepted: date

Abstract The improvement of data storage technology and data acquisition techniques in a variety of applications has led to accumulate huge data volumes. International research enterprises such as Human Genome Project and Digital Sky Survey are generating massive volumes of scientific data. A major challenge with these datasets is to glean insights from them. For instance, discovering structure, patterns, or originate relationships. The analysis of these massive, typically messy and inconsistent, volumes of data is indeed crucial and challenging in many application domains.

Today, data visualizations research community has introduced a number of visualizations tools to help analysts in exploring the information space to extract potentially useful information. However, when working with high-dimensional datasets, identifying visualizations that show interesting variations and trends in data is non-trivial: the analyst must manually specify a large number of visualizations, explores relationships among various attributes, and examine different subsets of data before discovering visualizations that are interesting or insightful. Current visualization tools provide various techniques to measure the *interestingness* of data such as computing deviations among data distributions, mining outlier aspects, and computing data variance. Those tools identify the interesting visualizations by exploring all possible visualizations in datasets. However, exploring all possible visualizations is a costly and time consuming process when the dimensionality is high. Furthermore, the rapid growth of databases becomes multifaceted in their channels and di-

I. Ibrahim
School of Infornation Technology and Electrical Engineering, University of Queensland, Australia
E-mail: i.ibrahim@uq.edu.au

A. Albarak
E-mail: a.albarrak@uq.edu.au

Xue Li
E-mail: xueli@itee.uq.edu.au

mensionality thus, the transition from static analysis to real-time analytics represents a fundamental paradigm shift in the field of big data.

Motivated by this, we propose an efficient framework called *Realtime Scoring Engine* (*RtSEngine*) that assists the analyst to limit the exploration of visualizations for a specified number of visualizations and/or certain execution time quote then recommends a set of views that meets the analyst's budgets. *RtSEngine* incorporates our proposed approaches to prioritize attributes that form all possible visualizations in a dataset based on their statistical properties such as selectivity, data distribution, and number of distinct values. Then, it recommends the views created from the top attributes. Moreover, we present visualizations cost-aware techniques that estimate the retrieval and computation costs of each visualization to recommend views while considering their costs. That is, the analyst may discard visualizations that take much time.

We have implemented a comparative experimental study to evaluate the effectiveness and efficiency of the proposed approaches running on different time and space limits. We asses the quality of visualizations and the overhead obtained by applying theses techniques on both synthetic and real datasets.

Keywords Query Visualization · Aggregate Queries · Visual Analytics

1 introduction

Data visualization is one of the most common tools for identifying trends and finding anomalies in big data. However, with high-dimensional datasets, identifying visualizations that effectively present interesting variations or patterns in the data is a non-trivial task: analysts typically build a large number of visualizations optimizing for a range of visualization types, appealing features, and more before arriving at one that shows something valuable. For datasets with large number of dimension attributes, it is extremely exhaustive for analysts to manually study all the attributes; hence, interactive visualization needs to be boosted with automated visualization techniques. Interactive visualization analytics tools such as *Tableau, ShowMe, and Fusion Tables* [9,21,28] provide some features for automatically recommending the best visualization for a dataset. However, these features are restricted to a set of aesthetic rules that guide which visualization is most appropriate.

Other tools such as *Profiler* [15] are exploring all data and the visualizations space to detects anomalies in data and provide some visualization recommendation functionality. Although, those tools are limited only to determine the best binning for the for the horizontal axis. Profiler maintains a data cube in memory and uses it to support rapid user interactions. While this approach is possible when the dimensionality and cardinality are small, it cannot be used with large tables and ad-hoc queries.

Another example of tools that recommend visualizations is *VizDeck* [16]. VizDeck recommends visualizations based on the statistical properties of small datasets and adopts a card game metaphor to help organize the recommended

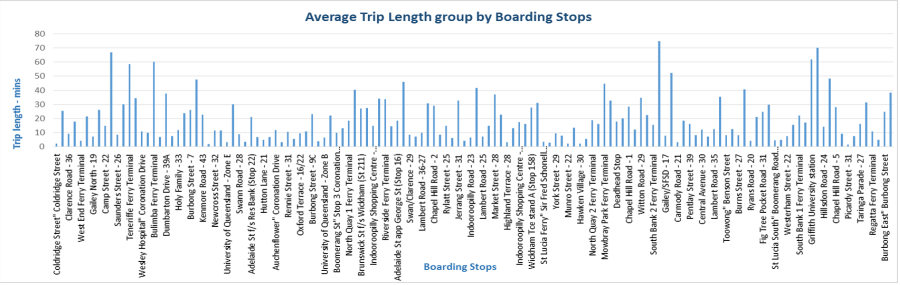


Fig. 1: Average trip lengths in minutes by boarding stop towards *University of Queensland*.

Average trip length in minutes by boarding stop

Boarding stop	Trip length
Macarthur Ave : Northshore Hamilton	74.85
Apollo Ferry Terminal	70.27
Bretts Wharf Ferry Terminal	67.01
Griffith University station	61.83
Bulimba Ferry Terminal	60.13

visualizations into interactive visual dashboard. VizDeck does not discuss techniques to efficiently generate these visualizations.

For large scale datasets, *SeeDB* [30] was proposed to automatically recommend interesting visualizations based on distance metrics which compute deviations among the probability distributions of the visualizations. SeeDB presents different levels of optimizations to decrease the latency and maintain the quality of visualizations such as sharing computations and combined query executions.

Although, these analytic tools present various approaches and measures to asses the interestingness of data, they still have to explore all possible visualizations to recommend a subset of interesting visualizations. Exploring the entire data and all visualizations space is almost impossible with the limited time and resources, especially when data is growing in both the dimensionality and cardinality. As a result, shifting from static analytics to realtime analytics is essential because of the rapid data accumulation when compared with a constant human cognitive capacity. Indeed this is a challenging problem. An interactive query visualizations tool needs to explore the data space intelligently by discounting unnecessary visualizations and recommending only essential visualizations while preserving the quality of the results.

The following example discuss the main idea to identify the useful and interesting visualization process workflow.

Example 1 Consider a transportation analytics team that is undertaking a study for a particular alighting stop: *University of Queensland* (UQ). This stop has poor performance and has received a lot of passengers complaints.

Suppose that the team uses the *GoCard* database containing metrics such as trip length and daily passengers for each route. Also, it has a large set of dimension attributes containing information such as operators, service number, operations date, day, direction, boarding stop, alighting stop, etc.

Given the large size of the database (millions of records), an analyst will overwhelmingly use a collection of visualization programs to gather insights into the behavior of alighting stop UQ. In a typical analysis workflow, an analyst would begin by using the program's GUI or a custom query language to execute the equivalent of the following SQL query and pull all data from the database for the alighting stop UQ:

```
Q = SELECT * FROM GoCard WHERE alighting stop ="University of
                              Queensland";
```

Next, the analyst would use an interactive GUI interface to visualize various metrics of the query result. For instance, the analyst may visualize average trip length grouped by route, total daily passengers grouped by direction, maximum trip length by boarding stop, and so on. Under the hood, these visualization operations are essentially queries to the underlying data store and subsequent graphing of the results. For instance, the visualization for average trip length by boarding stop is generated by running an operation equivalent to the SQL query Q_1 shown below. Table **??** and Figure 1 show an example of Q_1's result and a potential visualization.

```
Q1 = SELECT boarding stop, AVG(trip length) FROM GoCard WHERE
alighting stop ="University of Queensland" GROUP BY boarding
                                  stop;
```

□

Example 1 above suggests that visualizations that portray trend deviations from a reference are potentially remarkable to analysts. SeeDB is a data querying and visualization tool that allows analysts to manually generate their own visualizations, or get data-driven recommendations on demand based on the deviation in data. Specifically, it implements a deviation-based utility metric to identify the most interesting visualizations from a large set of potential visualizations. Then, it posits that a visualization is potentially *interesting* if it shows a trend in the subset of data selected by the analyst (i.e., metrics about UQ) that deviates from the equivalent trend in the overall dataset. In our Example 1, the average trip lengths grouped by boarding stops (Figure 1) is considered as the top interesting visualization produced by SeeDB. The reason is, it depicts long averages trips in some boarding stops which travels towards UQ that are significantly different from the equivalent average of the trip lengths (equals 17.6 minutes) in the entire dataset.

As listed in Table 1, ferry terminals scored longer trips to UQ than bus stops because ferries often take longer waiting times among stops than buses. However, some bus stops obtained long trips according to the geographical location or the distance such as *Griffith University station*.

We summarize our contributions as follows:

- Proposing a new problem which address the limitation of current visualizations recommendation tools. Particularly, we include budget constraints to automatically recommend top-K interesting visualizations according to an input query within the specified budget.
- Designing an efficient framework called *Realtime Scoring Engine* (*RtSEngine*) that limits the exploration of visualizations by assessing priorities of the recommended views according to their deviation utilities and costs.
- Proposing efficient algorithms which utilize statistical features of the views such as number of distinct values, selectivity ratios, and data distribution, to early prioritize the views.
- Proposing efficient algorithms to approximate the retrieval and computations costs of the generated visualizations and evaluate their estimated costs against their deviation utilities to recommend high accuracy views in the specified budgets.
- Conducting extensive experiments that demonstrate the efficiency and effectiveness of our proposed algorithms on real and synthetic data set.

This paper is organized as follows: Section 2 provides preliminary details on recommendation of query visualizations and present our problem statement. Then, we present our framework *RtSEngine* in Section 3 that contains two main modules: Priority Evaluator and Cost Estimator, which recommend a set of visualizations efficiently within the specified constraints. Section 4 shows experiment results for our proposed algorithms on two real datasets. Finally Section 5 describes related works on query visualization.

2 Background and Preliminaries

In this section, we present some background details about visualizations in the context of structural databases. We start by explaining how a visualization is constructed by an SQL query, and how to measure the interestingness of a visualization based on a model proposed by [30]. Then, we formally present our problem statement.

A visualization V_i is constructed by an SQL select-project-join query with a group-by clause over a database D. The dimensions in a database table are classified into two sets: dimension attributes set $A = \{a_1, a_2, ...\}$, and measure attributes set $M = \{m_1, m_2, ...\}$. While the set $F = \{f_1, f_2, ...\}$ contains all aggregation functions. Hence, each visualization V_i is represented as a triple (a, m, f), where a is an attribute applied to the aggregation function f on a measure attribute m.

As an example, $V_i(D)$ visualizes the results of grouping the data in D by a, and then aggregating the m values using f. This view is called the *reference view*. Consequently, $V_i(DQ)$ represents a similar visualization applied to the result set denoted as DQ for a given user query Q, and is called the *target view*. An example of a target view is shown in Figure 1 where a is the boarding stops, m is the trip length, and f is the average aggregation function.

Any combination of (a, m, f) represents a view. Accordingly, we can define the total number of possible views as follows:

$$\text{View Space}(SP) = 2 \times |A| \times |M| \times |F| \tag{1}$$

Though, in the context of Big Data, SP is a very large number. Hence, there is a need to calculate the *utility* of all those SP views so that exploring them become practical. To do that, the utility of a visualization is measured as its *deviation* from a reference dataset D_R. For instance, visualizations that show different trends in the query dataset (i.e. DQ) compared to a reference dataset D_R are supposed to have high utility. The reference dataset D_R may be defined as the entire underlying dataset D, the complement of $DQ(D-DQ)$ or data selected by any arbitrary query $Q'(DQ')$.

Given an aggregate view V_i and a probability distribution for a target view $P(V_i(DQ))$ and a reference view $P(V_i(D_R))$, the utility of V_i is the distance between these two normalized probability distributions. The higher the distance between the two distributions, the more likely the visualization is to be interesting and therefore higher the utility. Formally:

$$U(V_i) = S(P(V_i(DQ)), P(V_i(D))) \tag{2}$$

Where S is a distance function (e.g., Euclidean distance, Earth Movers distance, etc). Hence, the problem of visualizations recommendation is as follows [30]:

Definition 1 Given a user-specified query Q on a database D, a reference dataset D_R, a utility function $U()$, and a positive integer K. Find K aggregate views $V_1, V_2, ..., V_K$ that have the highest utilities among all views while minimizing total computation time.

Now, we are in place to present our problem statement for visualization recommendations.

2.1 Problem Statement

Our proposed problem statement for visualization recommendations incorporates two limits (i.e., input parameters) to overcome the limitation of exploring all views.

Definition 2 Given a user-specified query Q on a database D, a reference dataset D_R, a utility function $U()$, a positive integer K, an execution time limit tl and a views number limit R where $K \leq R \leq SP$. Find K aggregate views $V \equiv (a, m, f)$ which have maximum utilities $U(V)$ among all possible views in the specified limits R and tl while maximizing the accuracy among all Top-K views chosen from all SP views.

The limits tl and R in Def. 2 are added explicitly to overcome the limitation of exploring all views. The former is a time budget that any algorithm should not exceed, while the latter is an upper bound on the number of views to be explored. For instance, tl can be set to zero, and $R = SP$. That is, no limit on the execution time and no limit on the number of generated views.

While those limits can be tuned by any valid value, an algorithm should output the same views as if there were no limits. This requirement makes the problem non-trivial, hence, we address it by presenting our optimization techniques encapsulated within the *RtSEngine* framework.

3 Methodology: *RtSEngine* Framework

The goal of *RtSEngine* is to recommend a set of aggregate views that are considered interesting because of their abnormal deviations. To achieve that, *RtSEngine* utilizes the following key idea: recommend views that are created from grouping high ranked dimension attributes A' within the set A. The attributes ranks in A' are computed using our proposed prioritizing techniques discussed later in the following sections. Essentially, those techniques evaluate the priorities of all dimension attributes according to their statistical features gathered from the meta-data, e.g., number of selected values, data distribution, and selectivity. Then, by reordering all dimension attributes according to their priorities, only a subset of high priority attributes are passed to the execution engine, hence, limiting the number of examined views and execution time.

Conceptually, *RtSEngine* is designed as a recommendation plug-in that can be applied to any visualization engine, e.g., Tableau and Spotfire. However, in this work, we built *RtSEngine* as a standalone end-to-end system on top of SeeDB which allows users to pose arbitrary queries over data and obtain recommended visualizations. *RtSEngine* is comprised of two main modules (See Figure 2):

1. **Priority Evaluator:** An underlying module in front of any recommendation engine. Used to evaluate the dimension attributes that form visualizations according to a priority function Pr computed using our proposed techniques.
2. **Cost Estimator:** This module is supposed to run in parallel with the Priority Evaluator to estimate the retrieval and computation costs of each visualization using our estimation approaches. Estimating the visualization costs in real-time improves the efficiency by discounting high costs and low priorities visualizations. Note that this module is an awareness cost approach which incorporates the estimated costs to assess visualizations based on their priorities and costs.

We define a notion of benefits $Benefit(V_i)$ of a view V_i as the gains from each view represented as the utility of view $U(V_i)$, compared with the time spent $Cost(V_i)$ to compute the view V_i. Formally:

$$Benefit(V_i) = \frac{U(V_i)}{Cost(V_i)} \tag{3}$$

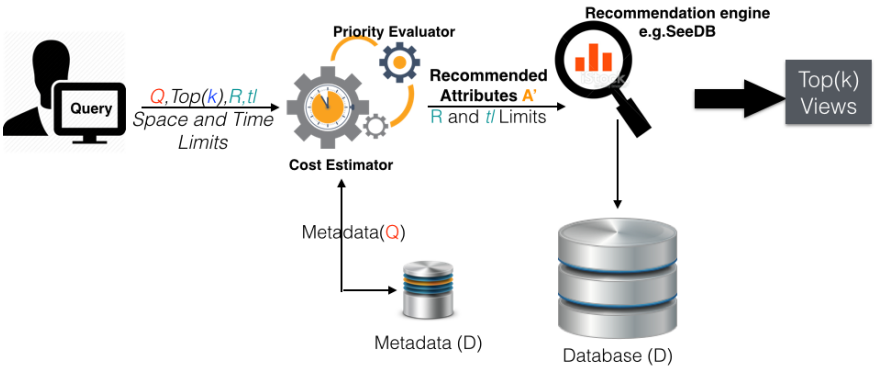


Fig. 2: *RtSEngine*: Real time Evaluation Architecture for Automatic recommendation

Cost estimations of visualizations is discussed later on Section 3.2. Both modules (Priority Evaluator and Cost Estimator) read information by querying metadata to collect information about dimension attributes, e.g., number of distinct values and cardinality. Next, we describe the two modules in details.

3.1 Priority Evaluator: Dimension Attributes Prioritizing

In this section, we discuss the proposed approaches for prioritizing the dimension attributes in the both results set DQ and reference set (e.g. the entire dataset D) and suggest a set of visualizations that are likely to be interesting and score high deviation utilities in certain realtime limits such as maximum number of explored visualizations and execution time. The proposed approaches are based on our observations about the difference between the number of distinct values in the dimension attribute in the results set DQ and the entire dataset D affects on the deviation measures. In addition, other statistical features may also affect such as data distribution and selectivity; such features will be discussed in more detail in the next subsections. The following example illustrates this observation and describes how our strategies are agnostic for any recommendation system.

Example 2 Suppose a flights database keeps flights records which contains two dimension attributes such as destination airport name and airlines and one metric is arrival delays. Given the large size of the database (millions of records) contains 100 airports and 20 airlines companies, the analyst will study the average delays visualizations grouped by airports and airlines using a recommendation tool e.g. SeeDB and comparing these views with a reference set to glean insights about all flights departure from $Origin1$. These views can be expressed as SQL queries:

– V_1: select AVG (arrival delays), airport from flights where origin='Origin1' group by airport;

– V_2: select AVG (arrival delays), airlines from flights where origin='Origin1' group by airlines;

□

For instance, both visualizations V_1 and V_2 in Example 2 have the same number of distinct values: 10 destinations airports and 10 airlines operators. Eventually, aggregate views V_1 and V_2 will be compared to the corresponding reference views (i.e., the entire dataset D) according to a metric. In [30] for instance, it uses a deviation-based metric that calculates the distance between the normalized distributions between the target and reference views. In our Example 2, the average arrival delays of 10 destinations airports in view V_1 are evaluated against the average arrival delays of 100 destinations airports in the entire dataset D. Similarly, the average arrival delays of the 10 airlines operators in view V_2 are compared against the average arrival delays of the all 20 airlines operators in the entire dataset D. Thus, only 10 distinct values in view V_1 will be compared with equivalent values in the reference view, while the remaining 90 distinct values would have no equivalent values in the target view. As a result, those remaining 90 distinct values will be compared with zeros.

Furthermore, in view V_2 there are only 10 airlines operators that would be compared with zeros. This illustration arises a question about the impact of the difference in distinct values of views and their data deviations according to distance-based metrics. Formally, $Dval(V_i(DQ))$ is defined as the number of distinct values in a target view V_i. Consequently, $Dval(V_i(D))$ is the number of distinct values in the corresponding reference view V_i. In Example 2, $Dval(V_1(DQ)) = 10$ and $Dval(V_1(D)) = 100$. As mentioned previously, the deviation of each visualization is captured by a distance based metric that computes the distance between two probability distributions of views. That is, the deviation of a visualization V_i is its utility defined in Eq. 2: $U(V_i) = S(P(V_i(DQ)), P(V_i(D)))$. The distance metric $S()$ is a distance function such as Euclidean, Earth-Mover distance, ... , etc.

We discuss the influence of the difference in distinct values on computing the view utility $U(V_i)$ using Euclidean distance (although our experiments are using Earth Movers distance function as the default deviation measure). As shown in Eq. 4, L_2-norm distance evaluates all aggregated values (points) in both views $V_i(DQ)$ and $V_i(D)$ to find the utility $U(V_i)$. Hence, V_1's utility in Example 2 is obtained by computing the L_2-norm distance between the average arrival delays (values) of destination airports (points) in $V_1(DQ)$ and all airports in $V_1(D)$ the entire dataset. Formally:

$$U(V_i) = \sqrt{\sum_{j=1}^{n} (V_iD(y_j) - V_iDQ(x_j))^2} \tag{4}$$

where $n > 0$ is the maximum number of points among V_iD and V_iDQ. Since the compared views (i.e. target and reference view) may contain different number of distinct values, we denote n' as the number of records in $V_i(DQ)$ and n'' as the number of records in $V_i(D)$. Hence, we can rewrite the utility equation

Algorithm 1: $Diff_DVal$

Input: Attributes $A(a_1, a_2, ..., a_n)$, Query Q, Views limit R
Output: Set $\mathcal{H}$: Highest priorities of dimension attributes
$\mathcal{C} = \phi$ Set of all dimension attributes priorities ;
for $i = 1$ ***to*** n **do**
  $DvalD(a_i) \leftarrow$ number of distinct values of a_i in D ;
  $DvalDQ(a_i) \leftarrow$ number of distinct values of a_i in DQ ;
  $Pr(a_i) = |DvalD(a_i) - DvalDQ(a_i)|$;
  $\mathcal{C} \leftarrow Pr(a_i)$;
Sort $\mathcal{C}$;
$G = |\frac{R}{M \times F}|$ Calculate the required dimension number;
for $i = 1$ ***to*** G **do**
  $\mathcal{H} \leftarrow \mathcal{C}.get(i)$;
return $\mathcal{H}$;

of view V_i as follows:

$$U(V_i) = \sqrt{\sum_{j=1}^{n'} (V_iD(y_j) - V_iDQ(x_j))^2 + \sum_{j=n'+1}^{n''} (V_iD(y_j) - 0)^2} \qquad (5)$$

where $n' < n''$, and $n = n' + n''$. Because there are only n' values in the target view V_iDQ, then all subsequent points in the reference view V_iD, i.e., $n'' - n'$ values, would be compared with zeros. The higher the difference between distinct values in corresponding views forces much remaining values to be compared with zeros and increases the distance among views. In Example 2, the number of records n' of both target views $V_1(DQ)$ and $V_2(DQ)$ equals 10. However, the number of records in the reference views, i.e., n'' are $V_1(D) = 100$ and $V_2(D) = 20$. V_1 is expected to show higher distance (deviation) than V_2 when computing L_2 norm distance because 90 airports would be evaluated to zeros in V_1 but there are only 10 airlines operators with zero values in view V_2. Since, every view is an aggregate group by query over a dimension attribute as described earlier, then, the number of records in each view equals the number of distinct values in the grouped dimension attribute.

Such observations can be utilized to early asses these (visualizations) views before executing the underlying queries to avoid computational costs (i.e., retrieval and deviation measure costs) by evaluating dimension attributes that contribute in creating visualizations. Furthermore, evaluating dimension attributes can also be done using other statistical properties such as selectivity and data distribution.

Further discussion of utilizing these features in our proposed approaches is presented in the next sections.

3.1.1 Ranking Dimension Attributes based on Distinct Values

Scoring dimensions based on difference of distinct values is the first class of prioritizing algorithms. This approach is referred to as $Diff_DVal$, and it is

Algorithm 2: *Sela*

Input: Attributes $A(a_1, a_2, ..., a_n)$, Query Q, Views limit R
Output: Set $\mathcal{H}$: Highest priorities of dimension attributes
$\mathcal{C} = \phi$ Set of all dimension attributes priorities ;
for $i = 1$ ***to*** n **do**
  $DvalD(a_i) \leftarrow$ number of distinct values of a_i in D ;
  $DvalDQ(a_i) \leftarrow$ number of distinct values of a_i in DQ ;
  $Pr(a_i) = DvalDQ(a_i) * Sel_{a_i}^{DQ} + (\frac{DvalDQ(a_i)}{DvalD(a_i)}) * Sel_{a_i}^{D}$;
  $\mathcal{C} \leftarrow Pr(a_i)$;
Sort $\mathcal{C}$;
$G = |\frac{R}{M \times F}|$ Calculate the required dimension number;
for $i = 1$ ***to*** G **do**
  $\mathcal{H} \leftarrow \mathcal{C}.get(i)$;
return $\mathcal{H}$;

based on the basic observation about the number of distinct values of the dimension attributes in the results set DQ and the entire database D. The $Diff_DVal$ algorithm scores the dimension attributes according to the difference between the normalized distinct values of attributes in the result set DQ and the entire database D. Algorithm 1 inputs a query Q, a set of dimension attributes A, maximum views limit R, and/or execution time limit tl. Then $Diff_DVal$ obtains the number of distinct values for all dimension attributes in both results sets DQ and reference dataset D by posing underlying queries to select the count of distinct values. After getting the number of distinct values, $Diff_DVal$ computes the priority of each dimension attribute as the difference between each normalized values. Then, $Diff_DVal$ sorts all dimension attributes based on their priorities. Based on Eq. 1, $Diff_DVal$ computes the required number of dimension attributes G that creates the limit number of views R, then it returns the set $\mathcal{H}$ of size G that contains a group of high priorities attributes.

In case there is an execution time limit tl, $Diff_DVal$ returns an ordered set of all dimension attributes based on their priorities, and then it passes the time limit tl to the recommendation visualization engine to limit the executions.

3.1.2 Scoring Dimension Attributes based on Selectivity

In this section, we discuss another variation of scoring the dimension attributes by capturing the data distribution in terms of query size and selectivity. Selectivity estimation is at the heart of several important database tasks. It is essential in the accurate estimation of query costs, and allows a query optimizer to characterize good query execution plans from unnecessary ones. It is also important in data reduction techniques such as in computing approximated answers to queries [1,8]. Databases have relied on selectivity estimation methods to generate fast estimates for result sizes [5,3,22,23].

The selectivity ratio [17] is defined as follows:

Definition 3 The degree to which one value can be differentiated within a wider group of similar values.

The selectivity ratio also known as the number of distinct unique values in a column divided by its cardinality [16]. Formally, the selectivity ratio of attribute a_i is:

$$Sel_{a_i}^{B} = \frac{\text{Number of distinct values of } a_i \text{ in } B}{\text{Cardinality of } a_i \text{ in } B} \tag{6}$$

where B is either the result set DQ or the reference dataset D, and $0 < Sel_{a_i}^{B} \leq 1$.

For the flight database in Example 2, both the result set DQ and the reference set D have a fixed number of records, which reveals that the selectivity ratio of the airlines column is usually low because we cannot do much filtering with just the 20 values. In contrast, the selectivity ratio of the airports column is high since it has a lot of unique values. Our proposed approach $Sela$ utilizes the number of distinct values in the dimensions attributes and incorporates the query size to identify priorities of these dimensions by calculating a priority function $Pr()$ for each dimension attribute. Then $Sela$ reorders the dimension attributes based on the priority.

Using selectivity ratio and the number of distinct values for assessing visualizations in D and DQ gives closer insights about the data characteristics such as the size (number of records) of aggregated views generated from group by attributes and the uniqueness degree of data in each dimension attribute. Again, in the flights database Example 2, DQ have 10 distinct airports out of 100 airports in the airports column. This means any visualization constructed by grouping airports column in result set DQ contains only 10 aggregated records. Hence, using the query size assists on quantifying how many records would be aggregated in each view that formed from grouping a dimension attribute. However, capturing the change of both number in distinct values and the number of aggregated records in each dimension attribute in result set DQ and reference set D is essential to identify visualizations that produce high deviations among all possible visualizations. Thus, we modified the priority function $Pr()$ in $Sela$ to consider the number of records in each dimension attribute a_i and its selectivity ratio. Formally:

$$Pr(a_i) = DvalDQ(a_i) * Sel_{a_i}^{DQ} + (\frac{DvalDQ(a_i)}{DvalD(a_i)}) * Sel_{a_i}^{D} \tag{7}$$

The attribute priority $Pr(a_i)$ evaluates the number of distinct values for each dimension attribute in result set DQ multiplied by its selectivity ratio. This identifies the distinct values variations and the diversity through each dimension attribute when compared with the number of records. Furthermore, the same number of distinct values is assessed in the corresponding dimension attribute of the reference set D while considering the number of records.

In $Sela$, high priority dimension attributes are assumed to produce aggregate views (i.e., target views) that contain many groups (i.e., points) which

Algorithm 3: *DimsHisto*

Input: Attributes $A(a_1, a_2, ..., a_n)$, Query Q, Views limit R
Output: Set $\mathcal{H}$: Highest priorities of dimension attributes
$\mathcal{C} = \phi$ Set of all dimension attributes priorities ;
for $i = 1$ ***to*** n **do**
  Compute $HD(a_i)$ and $HDQ(a_i)$;
  $Pr(a_i) = d(HD(a_i), HDQ(a_i))$;
  $\mathcal{C} \leftarrow Pr(a_i)$;
Sort $\mathcal{C}$;
$G = |\frac{R}{M \times F}|$ Calculate the required dimension number;
for $i = 1$ ***to*** G **do**
  $\mathcal{H} \leftarrow \mathcal{C}.get(i)$;
return $\mathcal{H}$;

are aggregated from records in the result set DQ. Also, the same high priority dimension attributes are assumed to produce aggregate views (reference views) by aggregating larger number of records in reference set D. This has direct effect on the aggregated values and the number of groups in both target and reference views which is expected to score high deviation utilities.

Although the $DiffDval$ approach prioritizes dimension attributes (aggregate views) according to the number of distinct values, it is limited since it is incompetent to prioritize dimension attributes (aggregate views) when the number of distinct values remains stable in both result set DQ and reference set D. Moreover, $DiffDval$ does not consider the data distribution within the attributes. To overcome this limitation, $Sela$ utilizes the number of records and the selectivity ratios of dimension attributes in both datasets DQ and D.

The proposed algorithm $Sela$ firstly computes the priority of each dimension attribute based on Eq. 7. Then, it sorts the dimension attributes based on the assigned priority to create a set $\mathcal{H}$ of the top G dimension attributes. In case of execution time limit tl, $Sela$ returns an ordered set of attributes with the highest priorities and passes time limit tl to the recommendation visualization engine to limit the executions.

3.1.3 Prioritizing Dimension Attributes based on Histograms

We proposed $Sela$ and $Diff_DVal$ approaches to automatically recommend views with the highest deviations based on a priority for each dimension attributes in a star schema database D. Specifically, the proposed approaches relay on the number of the distinct values and the selectivity ratio of each dimension attribute in the compared datasets (i.e., DQ and D) to compute the attributes priorities.

However, the limitation of the proposed approaches is using the selectivity ratio to reflect the degree of variations of data in the dimension attributes while ignoring the distribution of data itself. In addition, it is difficult to prioritize dimensions that have the same distinct values or the same selectivity ratio.

Hence, we propose the *DimsHisto* approach which attempts to capture data distribution inside the dimensions attributes by creating frequency histograms and directly measuring the distance among corresponding histograms to evaluate these dimensions attributes. *DimsHisto* firstly generates frequency histograms for all dimensions attributes in each dataset. Then it computes the deviation in each dimension by calculating the normalized distances between each corresponding dimension attribute. For any star schema database D, a dimension attribute $a_i \in A = \{a_1, a_2, ..., a_n\}$ can be represented as two frequency histograms: $H_{D(a_i)}$, and $H_{DQ(a_i)}$. Those two histograms are created by executing the following queries:
$H_{D(a_i)}$: Select $count(a_i)$ from D group by a_i;
$H_{DQ(a_i)}$: Select $count(a_i)$ from DQ group by a_i;
Then, after normalizing these histograms, the priority of each dimension attribute is computed as the distance between these two histograms:

$$Pr(a_i) = S(H_{D(a_i)}, H_{DQ(a_i)}) \tag{8}$$

Where $S()$ is a distance metric. Eventually, the dimension attributes are sorted according to their priorities.

A constructed histogram $H_{D(a_i)}$ is equivalent to all aggregate views created by aggregating any measure attribute (using aggregate function *Count*) and grouped by the dimension attribute a_i in the dataset D. Such a histogram assists in improving the performance of recommendation engines by avoiding the construction and computation of aggregate views along all measure attributes.

DimsHisto has to submit $2 \times |A|$ queries to compute the histograms of all dimensions and the computations of the distance metric. However, this step can be optimized to only $|A|$ by computing the histograms of all dimensions for the entire database offline. While *DimsHisto* can use any distance metric to compute the deviation among the views, we suggest to use the same metric to unify the metric of the deviations.

All proposed algorithms $Diff_DVal$, *Sela* and *DimsHisto* have the same number of queries as the cost of retrieving data. While *DimsHisto* has additional cost for distance computations, it shows high accuracy for most of the aggregate functions such as Sum, Avg, and Count, because these functions are relative to the data frequencies. Though, *DimsHisto* is less descriptive to other aggregate functions such as Min and Max, as they are not amenable for sampling-based optimizations.

3.2 Cost Estimator: Visualizations Cost Estimation

The previous approaches rank dimension attributes according to their priorities and recommend visualizations while being oblivious to the retrieval and computational costs of those visualizations. However, visualizations created using different dimension and measures attributes have different retrieval and execution costs according to the query size, type of the aggregate functions,

Algorithm 4: *ViewsEstimate*

Input: Attributes $A(a_1, a_2, ..., a_n)$, Measures $M(m_1, m_2, ..., m_o)$, Functions $F(f_1, f_2, ..., f_x)$
Output: $\mathcal{S}$: set of views estimated costs

```
for i = 1 to n do
    for j = 1 to o do
        for p = 1 to x do
            C(VDQ) = EstCostDQ(a_i, m_j, f_p);
            C(VD) = EstCostD(a_i, m_j, f_p);
            C(d(VDQ, VD)) Eq.10;
            Cost = C(VDQ) + C(VD) + C(d(VDQ, VD)) ;
            S.add(Cost, (a_i, m_j, f_p)) ;
return S;
```

number of groups in each attribute and the time used to compute the deviation among all values in the corresponding visualizations.

This urges the need to only generate visualizations with high deviations and avoid the computation costs of the low-deviation ones. Besides differences in deviation utilities among different visualizations, each visualization exhibits different execution and retrieval costs. Furthermore, some visualizations may take long computations and retrieval time to only yield small deviation distances. The trade-off between gaining high utilities of the visualizations and their computations and querying costs is challenging because it involves the optimizations of finding high utilities visualizations while considering their costs.

The cost estimation step is essential to determine the cost of running and computing the deviations of a visualization to evaluate its costs against the utility obtained by measuring the deviation among visualizations. To improve the performance of recommendation applications, it is vital to discard visualizations that are expected to consume much retrieval and computation time while returning low deviation distances.

The cost estimation modules approximate CPU and I/O costs to combine them into an overall metric that is used for comparing alternative plans. The problem of choosing an appropriate technique to determine CPU and I/O costs requires considerable care. An early study [20] identified key roles for accurate cost estimation, such as the physical and statistical properties of data. Cost models take into account relevant aspects of physical design, e.g., co-location of data and index pages. However, the ability to do accurate cost estimation and propagation of statistical information on data remains one of the difficult open issues in query optimization [4].

We determine the cost of a view $Cost(V_i)$ as the sum of the following:

- Cost of running view $V_i(a, m, f)$ on dataset D.
- Cost of running view $V_i(a, m, f)$ on dataset DQ.
- Computation cost of the distance function $S(V_iDQ, V_iD)$.

Formally:

$$Cost(V_i) = C(V_iDQ) + C(V_iD) + C(S(V_iDQ, V_iD)) \quad (9)$$

As mentioned previously, the cost of running a view $V_i(a, m, f)$ on a database is affected by various factors. For instance, access paths and indices that are used to execute the view determine the proper execution plan, which reflects the view execution cost.

Running Cost of Views $C(V_iDQ)$ and $C(V_iD)$: refers to the retrieval cost of the results of both views V_iDQ and V_iD as discussed earlier.

Computation Cost of $C(S(V_iDQ, V_iD))$: is considered as the time spent on calculating the distance measure $S()$ for each value in both corresponding views.

The number of points that are compared in the corresponding views V_iDQ and V_iD is the maximum number of groups (bins) among these two views, and it is denoted as n. Alternatively, it equals the maximum number of distinct values in V_iDQ and V_iD attribute dimension.

Note that the cost of distance measures vary according to their computational complexity. For example, the Euclidean distance is faster than the Earth Mover (EMD) distance function. This is because EMD has a very high complexity $O(n^3 logn)$ [14] while the complexity of the Euclidean distance is $O(n)$.

Since the computation cost depends on n and also depends on the computational complexity of the distance measure, we propose the following view cost equation:

$$C(S(V_iDQ, V_iD)) = O_d(n) \times d_t \quad (10)$$

Where O_d is the complexity of the distance measure, and d_t is the computation time used to compute a single point.

3.2.1 Retrieval Costs of Visualizations

In our context, the execution cost of views can be obtained using two different methods:

- **Actual Cost:** actual costs of the views are obtained by executing all queries to get their exact I/O costs, and calculating the deviation among the corresponding views.
- **DB Estimates:** reading the estimates of each view directly from the database engine (i.e., query optimizers).

However, our proposed cost method is not restricted to a certain cost estimation approach including methods based on sampling (e.g., [11,18]), histograms (e.g., [12]), and machine learning (e.g., [7,27]) which can be used to obtain the retrieval cost from independent estimation models.

Our proposed estimation algorithm $ViewsEstimate$ is illustrated in Alg.4. $ViewsEstimate$ takes dimensions, measures attributes, and the aggregated functions as input. Then it estimates I/O and computation time for each view V_i for both datasets and it returns the estimated costs of each view.

The estimated I/O time for each view is obtained by reading the estimation of queries from the database query optimizer or using an independent cost estimation model. Then, $ViewsEstimate$ calculates the computations costs of the distance measure between the corresponding views according to equation Eq. 10 to find the total estimated cost. Afterwards, $ViewsEstimate$ adds up the computations cost and the I/O cost for V_i, then stores it into set $\mathcal{S}$.

Cost Estimator utilizes the set $\mathcal{S}$ by defining a benefit of a dimension attribute $Benefit(a_i)$ as the priority of a_i divided by the maximum estimated cost of any view created using dimension attribute a_i, formally:

$$Benefit(a_i) = \frac{Pr(a_i)}{Cost(a_i)} \tag{11}$$

where $Cost(a_i)$ is the maximum estimated cost of any view created by grouping by a_i.

Finally, $DimsEstimate$ ranks dimension attributes depending upon their benefits as computed by Eq. 11. As shown in Alg.4, $ViewsEstimate$ inputs a set of dimensions and a visualization number limit R, then it iteratively calculates the priority and the cost of each dimension attribute to compute the benefit of each attribute. $ViewsEstimate$ computes the number of dimension attributes G that create the limit R, and then outputs a set of high $Benefit$ attributes of size G.

4 Experiments Results

We used two metrics for evaluating the results of the proposed approaches, one of these metrics is used by SeeDB to evaluate the quality of aggregate views [30]. Through the next set of experiments, we evaluate the impact of our proposed optimizations techniques.

Metrics: To evaluate the quality and correctness of the proposed algorithms, we used the following metrics:

- Accuracy: if $\{VS\}$ is the set of aggregate views with the highest utility, and $\{VT\}$ is the set of aggregate views returned by SeeDB baseline, then the accuracy is defined as:

$$Accuracy = \frac{1}{|VT|} * \sum x \text{ where } \begin{cases} x = 1 & \text{if } VT_i = VS_i \\ x = 0 & otherwise \end{cases}$$

  i.e., accuracy is the fraction of true positions in the aggregate views returned by SeeDB.
- distance-error: since multiple aggregate views can have similar utility values, we use utility distance as a measure of how far SeeDB results are from the true top-k aggregate views. Formally, SeeDB [29] defines distance error as the difference between the average utility of $\{VT\}$ and the average

utility of $\{VS\}$:

$$\text{distance-error} = \frac{1}{k}(\sum_i U(VT_i) - \sum_i U(VS_i))$$

4.1 Quality Evaluation Across Aggregate Functions

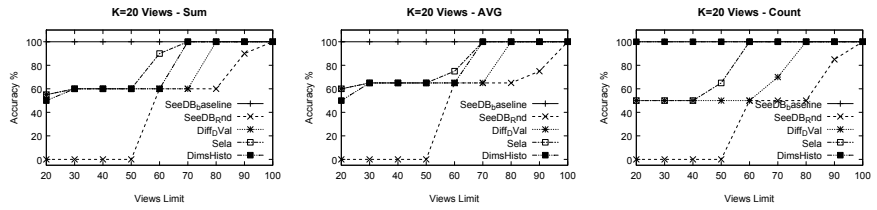
In these experiments we evaluate quality of the recommended visualizations produced by proposed techniques across different aggregate functions namely: *Count, Sum, Average, Min, and Max*. The dataset used is *Flight* database for flight delays in year 2008 obtained from The U.S. Department of Transportation's Bureau of Transportation Statistics (BTS) with size 250 K tuples [1]. The dataset contains 10 dimension attributes and 10 measures attributes. We run this experiment to assess the quality of the recommended views over each aggregate function *separately* with a space size(SP)=$1 \times 10 \times 10 = 100$ possible views and the deviation metric is Earth Movers Distance (EMD). All Experiments were executed 5 times and obtained the averaged results, we concerned with evaluating the the accuracy and utility of views produced by the proposed algorithms along limited number of views (referred as Views Limit) R and various sets of top deviated views K. In experiments, the analyst posed a query
Q : select * from ontime2008 where uniquecarrier ='American Airlines Inc.'
We implement two baseline strategies SeeDB baseline strategy processes the entire data and does not discard any views ($SeeDBbaseline$). It thus provides an upper bound on latency and accuracy and lower bound on error distance. The other baseline strategy we evaluate is the random strategy ($SeeDB_Rnd$) that returns a random set of k aggregate views as the result. This strategy gives a lower bound on accuracy and upper bound on error distance: for any technique to be useful, it must do significantly better than $SeeDB_Rnd$.

In the next experiments, we vary R the number of limited visualizations that explored denoted as *Views Limit* to recommend $K = 20$ visualizations and measure the accuracy, and error-distance for each of our strategies along different aggregate functions.

In summary, $Sela$ and $DimsHisto$ algorithms both produce results with accuracy $> 80\%$ and near-zero utility distance for all aggregate functions and a variety of R Views Limits particularly when Views Limits=60 views as shown in figures 3a,3b, and 3c. Moreover, they produce results with 100% accuracy and zero distance error after that limit. $Sela$ does slightly better than $DimsHisto$ as $Sela$ evaluates the recommended views by capturing the change of the selectivity ratios of dimension attributes that create views in both result set and reference set however, $DimsHisto$ scores accuracy 100% in figure 3c for aggregate function *Count* because the generated histograms from this algorithm are similar to the views created by counting dimension

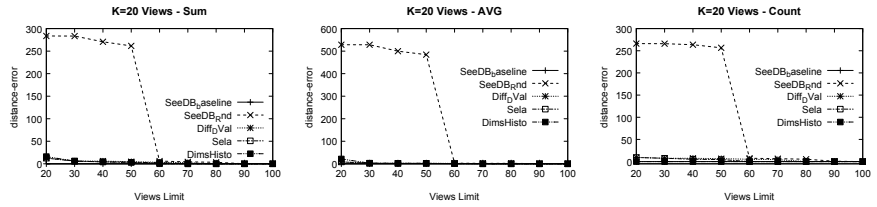
[1] http://www.transtats.bts.gov/

attribute values across different measure attributes. Algorithm $Diff_DVal$ is the lowest accuracy and the highest distance error among other algorithms specially for aggregate functions *Max and Min* as shown in figures 5a and 5b as it assess recommended views based on the difference of the distinct values only.



(a) Sum (b) Average (c) count

Fig. 3: Accuracy on varying view space sizes for the Algorithms *Sela* ,$Diff_DVal$, $DimsHisto$, and $SeeDB_Rnd$

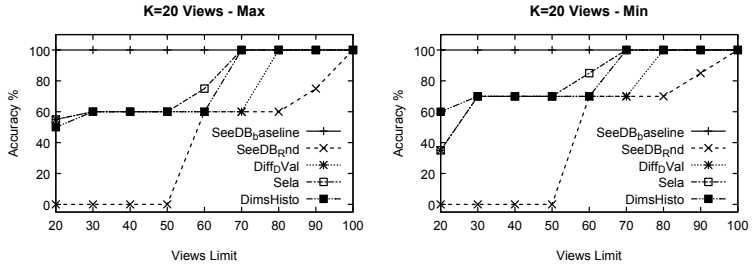


(a) Sum (b) Average (c) count

Fig. 4: Distance-error on varying view space sizes for the Algorithms *Sela* ,$Diff_DVal$, $DimsHisto$, and $SeeDB_Rnd$

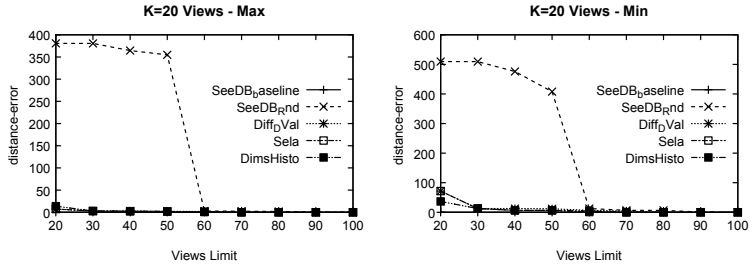
As shown in figures 4a,4b,4c The proposed algorithms produce results near-zero distance-error for all aggregate functions compared with lower baseline strategy $SeeDB_Rnd$ which produce views with low quality however, the quality of the recommended views produced by the proposed algorithms is almost near to the same utilities of views output by the top baseline $SeeDBbaseline$. The distance-error of results in the first view limits=20 and 30 views as shown in figures 6a,6b is high specially for the aggregate function *Min* because functions such as Min and Max are not docile for sampling but the proposed algorithms still score very low distance-error as shown in the figures.

In conclusion, the proposed techniques recommend high quality views in different views limits furthermore, the accuracy is increasing and it doesn't fluctuate along various views limits and similarly the distance-error is declining while increasing the number of explored views (views limit). In the worst



(a) Max (b) Min

Fig. 5: Accuracy on varying view space sizes for the Algorithms *Sela* ,$Diff_DVal$, *DimsHisto*, and $SeeDB_Rnd$



(a) Max (b) Min

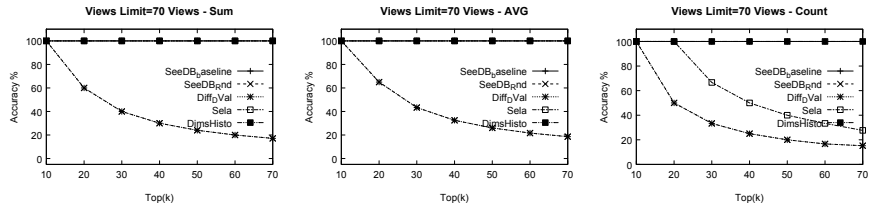
Fig. 6: Distance-error on varying view space sizes for the Algorithms *Sela* ,$Diff_DVal$, *DimsHisto*, and $SeeDB_Rnd$

cases, the accuracy and the distance-error remain fixed while increasing the number of explored views but they don't decrease.

In the following experiments, we vary K the number of visualizations to recommend and fix the number of explored visualizations as *Views Limit=70* and measure the accuracy, and error-distance for each of our strategies along different aggregate functions. We pay special attention to k = 10 and 20 because empirically these k values are used most commonly.

In summary, *Sela* and *DimsHisto* algorithms both produce results with accuracy 100% and zero distance-error when K=10 and 20 views for all aggregate functions as shown in figures 7a, 7b, 7c, 9a, and9b also algorithm $Diff_DVal$ scored accuracy 100% in the first number of recommended views K=10 views. Although, $Diff_DVal$ obtains the same accuracy as $SeeDB_Rnd$ for all aggregate functions but the $Diff_DVal$ scores much better distance-error than $SeeDB_Rnd$ as shown in figures 8a, 8b, 8c, 10a, and10b. As discussed in the previous experiment, the *DimsHisto* scores accuracy 100% specifically when the aggregate functions *Count* it is also succeeded to recommend views with 100% and zero distance-error for aggregate functions *Sum, Average, and*

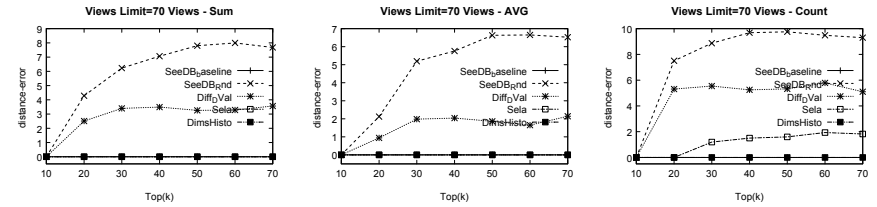
Count as shown in figures 7a, 7b, and 7c. In addition, we find that *Sela* and *DimsHisto* algorithms produce high quality views with 100% accuracy and zero distance-error for *Max* aggregate function , also they obtain $> 75\%$ and < 0.2 distance error for *Min* aggregate function when k=70 (Views Limit) as shown in 10a and 10b receptively.



(a) Sum (b) Average (c) count

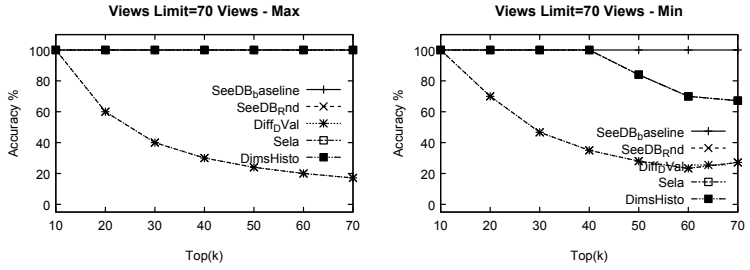
Fig. 7: Accuracy on varying view space sizes for the Algorithms *Sela* ,$Diff_DVal$, *DimsHisto*, and $SeeDB_Rnd$

The following figures describe the distance error for the proposed algorithms, we find that although $Diff_DVal$ approach score the same accuracy produced by $SeeDB_Rnd$ strategy but it obtains very low distance error along all aggregate function compared with $SeeDB_Rnd$ strategy as shown in figures 8a, 8b, 8c, 10a, and10b. To Sum up, the proposed approaches boast the accuracy of the recommended views for the mostly common used K values. Moreover, the *Sela* and *DimsHisto* achieve better quality results than $Diff_DVal$ because they are capturing the data distribution in the dimension attributes by using selectivity ratios and frequency histograms.



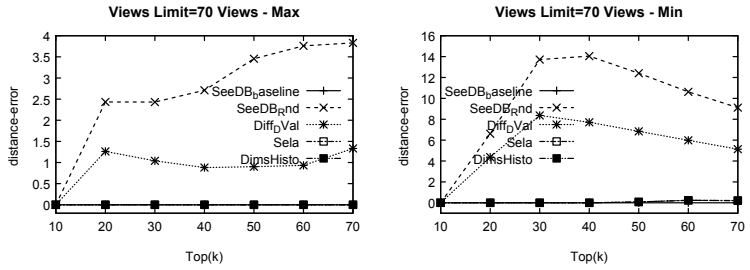
(a) Sum (b) Average (c) count

Fig. 8: Distance-error on varying k for the Algorithms *Sela* ,$Diff_DVal$, *DimsHisto*, and $SeeDB_Rnd$



(a) Max (b) Min

Fig. 9: Accuracy on varying K for the Algorithms $Sela$,$Diff_DVal$, $DimsHisto$, and $SeeDB_Rnd$



(a) Max (b) Min

Fig. 10: Distance-error on varying K for the Algorithms $Sela$,$Diff_DVal$, $DimsHisto$, and $SeeDB_Rnd$

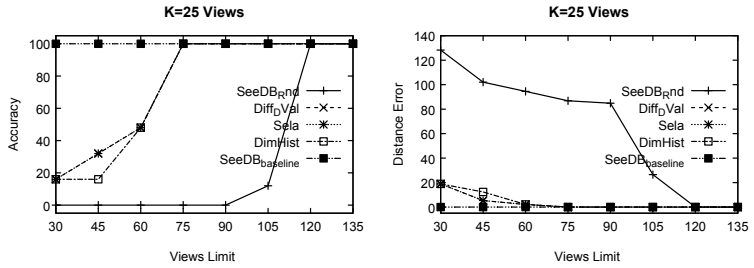
4.2 Accuracy evaluation

The dataset used is *GoCard* data with size 4.4 M tuples contains 9 dimension attributes and 3 measures attributes with we run this experiment on SeeDB over *All* aggregate functions *Count, Sum, Average, Min, and Max* and total of all possible views in each dataset referred as a space size=$5 \times 9 \times 3 = 135$ views and the deviation metric is Earth Movers Distance (EMD).

Q : Select * from gocard where alightingstop ='University of Queensland'

we implement two baseline strategies. The SeeDB baseline strategy processes the entire data and does not discard any views ($SeeDBbaseline$). It thus provides an upper bound on latency and accuracy and lower bound on error distance. The other baseline strategy we evaluate is the random strategy ($SeeDB_Rnd$) that returns a random set of k aggregate views as the result. This strategy gives a lower bound on accuracy and upper bound on error distance: for any technique to be useful, it must do significantly better than $SeeDB_Rnd$. In figure 11a shows the accuracy of the results produced by algorithms $Sela$, $Diff_DVal$, $DimHisto$, and $SeeDB_Rnd$ to find a top ($K = 25$) views comparing with different view space sizes. As shown all proposed algorithms $Sela$ and

$Diff_DVal$ scored the same accuracy in the first 30 explored views however, algorithm $DimsHisto$ shows lower accuracy than $Sela$ and $Diff_DVal$ when the number of explored views is 45 because $DimHisto$ evaluates dimension attributes according to their frequencies and it's less descriptive to some aggregate functions such as Max and Min. Thereafter, the proposed algorithms improve the accuracy to 100% and keep accuracy stable without any fluctuation. In addition, all proposed algorithms maintain the accuracy of results while growing with the extension of the space size while other algorithm the accuracy remains stable. Finally, as shown $SeeDB_Rnd$ is the lowest accuracy with different spaces except in the last two space limits.



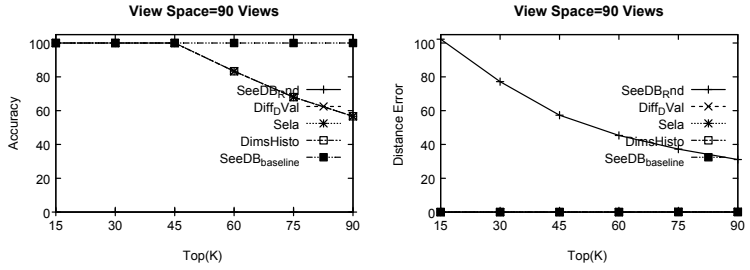
(a) Accuracy (b) Error Distance

Fig. 11: Results quality on varying view space sizes for the Algorithms $Sela$,$Diff_DVal$, $DimsHisto$, and $SeeDB_Rnd$

In figure 11b shows the distance error of the results produced by algorithms $Sela$, $Diff_DVal$, and $DimsHisto$ to find a top ($K = 25$) views across different number of explored views denoted as space sizes as shown algorithms succeeded to minimize the distance error quickly near to $SeeDBBaseline$ specifically when the expansion of the space sizes. Although, algorithm $DimsHisto$ obtained lower accuracy than $Sela$ and $Diff_DVal$ as shown in figure 11a at view space 45, but the distance error at the same view space is low because $DimsHisto$ recommended different views with high utility distances to minimize the distance error. Other algorithms succeeded to minimize the distance error quickly with expansion of the space sizes. $SeeDB_Rnd$ shows high distance error even if when the space size is large enough.

To sum up, the proposed algorithms evaluate the dimension attribute according to different priorities methods by recommending set of views which improve the quality of the view space limit R in terms of minimizing the distance error and enhancing the accuracy as shown in figures 11a and 11b.

In figure 12a shows the accuracy of the algorithms $Sela$, $Diff_DVal$, $DimsHisto$, and $SeeDB_Rnd$ in a fixed space size $R = 90$ views comparing with different view space top(k) views as shown all algorithms scored 100% accuracy in the first top 45 views which form half number of explored views. We observe that the accuracy declines while increasing top(K) in a fixed space



(a) Accuracy (b) Error Distance

Fig. 12: Results quality on varying top(K) views for the Algorithms $Sela$, $Diff_DVal$, $DimHisto$, and $SeeDB_Rnd$

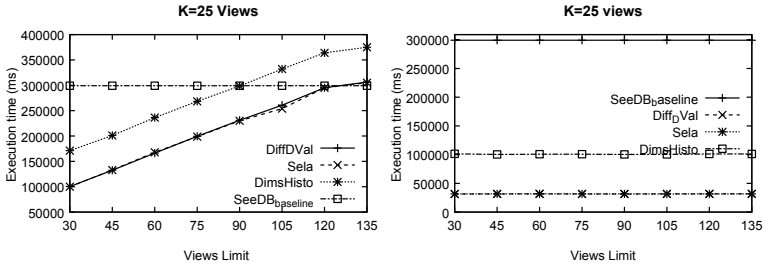
limit. This because wrong priortrizing of one dimension attribute than another will consequently affects on all recommended views that created from the wrong dimension attribute. However, the accuracy is above 50% when k=90 (the entire view space limit) as shown in figure 12a. Furthermore, the analyst is usually interested in recommending a small number of visualisations e.g. k=25.

In figure 12b shows the distance error of the algorithms $Sela$, $Diff_DVal$, $DimHisto$, and $SeeDB_Rnd$ in a fixed space size $R = 150$ views comparing with different Top(K) views all algorithms have a very small distance error ≈ 0 for Top(60) views however, $Diff_DVal$algorithm shows the smallest distance error across different K sizes. Both $Sela$ and $DimHisto$ report growing rise in the distance error with respect to Top(K) views required by the user in the certain view space size=90 views.

In conclusion, the discussed algorithms $Sela$,$Diff_DVal$, $DimHisto$ show high accuracy and low distance errors along different space sizes and varying Top(K) as illustrated previously however, these algorithms differentiate on the quality measures .For instance, algorithm $Sela$ and $Diff_DVal$ gain the highest accuracy than $DimHisto$ in the experiments as shown in figures 11a and 12a but algorithm $Diff_DVal$ has the lowest error distance as shown in figures 11b and 12b.

4.3 Efficiency evaluation

In this section, we evaluated the efficiency of the prioritizing algorithms in terms of the execution overhead added to SeeDB (as automatic recommendation engine) by computing the costs of executing the proposed algorithms and gains of applying the algorithms. Similarly, as the previous experiments, we measured the efficiency the algorithms by running experiments to capture the overhead and the gains along different $Top(K)$ and varying space limits compared with the actual execution of SeeDB baseline. All the experiments were



(a) Execution times of algorithms across views limits

(b) The overhead costs of proposed algorithms

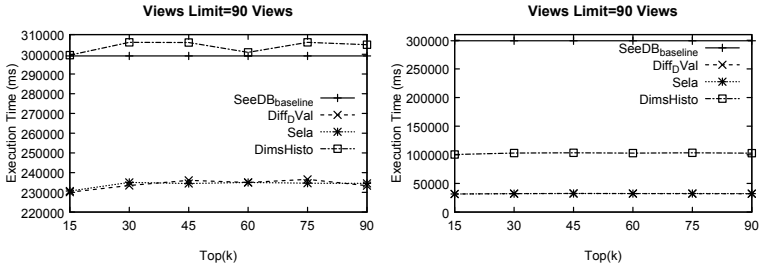
Fig. 13: Algorithms Performance on different space limits

repeated 5 times and the measurements were averaged. We begin by presenting a summary of our experimental findings and then dive into performance results for individual optimizations.

we compared the executions costs of the algorithms added to SeeDB with the original baseline of SeeDB to identify the improvements in the performance. As shown in figure 13a shows the total SeeDB and algorithms execution times compared with the original SeeDB baseline. This figure shows the improvements in the SeeDB performance with different space limits to find a top (K = 25) views. As shown the improvements in the performance by using the proposed algorithms are significant compared with the baseline furthermore, the execution costs increase linearly with the view space limit produced by algorithms.

In Figure 13b illustrates the executions times of the algorithms $Sela$, $Diff_DVal$ and $DimsHisto$ across different space limits however, this cost is considered as extra overhead should be added. As presented the executions times of the algorithms are almost stable along different space sizes because the algorithms evaluate a fixed set of dimension attributes every time. However, algorithm $DimsHisto$ is costly as it poses same number of queries to create histograms then it computes the distance among those histograms.

The following figures discuss the efficiency of the proposed algorithms along different $Top(K)$ views in a certain space limit=90. As shown in figure 14a, the proposed algorithms show improvements in the execution more than 40% compared with the SeeDB baseline execution time. As discussed earlier, algorithm $DimsHisto$ shows the highest cost among algorithms $Sela$ and $Diff_DVal$. In other hand, figure 14b describes the execution costs of the algorithms separately. The running costs of algorithms are fixed while expanding K the number of top deviated views because the algorithms run on a specified space limit.



(a) Total Execution time of the algorithms on fixed space limit

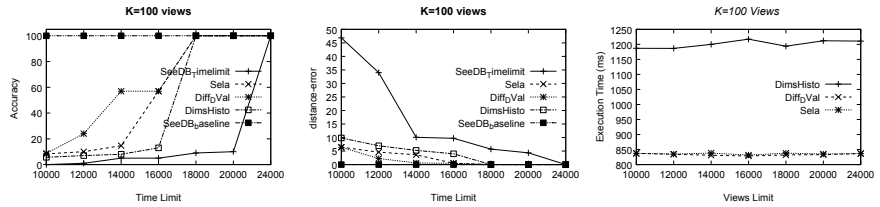
(b) Overhead costs of the algorithms on fixed space limit

Fig. 14: Algorithms Performance on varying $Top(k)$ views

4.4 Time Limit Experiments Results

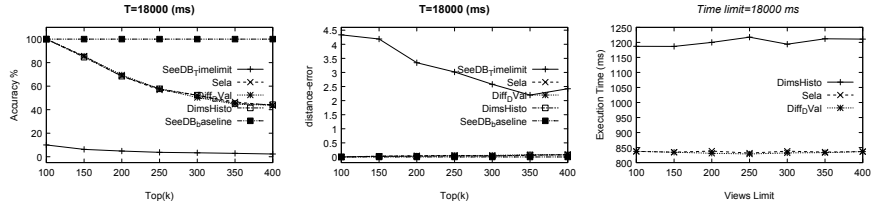
The dataset used is *Flight* database for flight delays in year 2008 obtained from The U.S. Department of Transportation's Bureau of Transportation Statistics (BTS) with size 250 K tuples [2]. The dataset contains 10 dimension attributes and 10 measures attributes with we run this experiment on SeeDB over *All* aggregate functions *Count, Sum, Average, Min, and Max* with a space size=$5\times 10\times 10 = 500$ views and the deviation metric is Earth Movers Distance (EMD). All Experiments were executed 5 times and obtained the averaged results, we concerned with evaluating the performance and the efficiency of the proposed algorithms along different time limits tl and various sets of top deviated views K. In experiments, the analyst posed a query
Q : select * from ontime2008 where dimmonth in('APR','MAY','JUN')
to compare the second quarter's dataset with the entire dataset and run to find different K views while varying time limits. In addition we evaluated the quality of the results of $top-K$ views produced by each algorithm with those produced by SeeDB baseline e.g. without time limits or any optimizations used and the average SeeDB baseline execution time is 23200ms. we implement $SeeDB_Timelimit$ strategy which processes the entire data and views in a specified execution time limit and it recommends top views that processed in that time limit. This strategy gives a lower bound on accuracy and upper bound on error distance: for any proposed technique should be useful.
In figures 15a and 15b show the accuracy and distance error of the results produced by algorithms $Sela$, $Diff_DVal$, $DimHisto$, and $SeeDB_Timelimit$ to find a top ($K = 100$) views compared with SeeDB baseline on different execution time limits. These algorithms outputs an ordered set of dimension attributed based on their priorities and submit the ordered set to execution engine $SeeDB_Timelimit$ then it processes all views generated according to the ordered set that produced by algorithms. As shown, $SeeDB_Timelimit$ shows high distance error and very low accuracy as well while the algorithms $Sela$ and

[2] http://www.transtats.bts.gov/



(a) Accuracy (b) Distance-error (c) Execution times

Fig. 15: Performance of Algorithms $Sela$, $Diff_DVal$, $DimHisto$, and $SeeDB_Timelimit$ on different time limits



(a) Accuracy (b) A Distance-error (c) Execution times

Fig. 16: Performance of Algorithms $Sela$, $Diff_DVal$, $DimHisto$, and $SeeDB_Timelimit$ on varying top(K) views

$Diff_DVal$ scored higher accuracy than $DimHisto$. Although the proposed algorithms show a growing in accuracy while extending the time limit but they achieved 100% accuracy from time limit 18000ms. The algorithms boasted the performance more than 30% and preserved the quality of views. In the other side, figure 15c describes the execution costs referred as overhead time of the proposed algorithms on the same experiment. $DimHisto$ algorithm execution time is about 1200ms while the algorithms $Sela$ and $N - N'$ have almost the similar execution time about 825ms. This shows that $Sela$ and $Diff_DVal$ algorithms are faster 66 % than $DimHisto$ algorithm. As discussed previously the additional histograms distance computations caused the extra overhead algorithm $DimHisto$.

In figures 16a and 16b show the accuracy of the algorithms $Sela$, $N - N'$, $DimHisto$, and $SeeDB_Timelimit$ in a certain time limit $tl = 18000$ ms comparing with different sets of top(k) views as shown all algorithms scored 100% accuracy in the first top 100 views. However, The accuracy declines with the increasing top(K) in a fixed time limit but the proposed algorithms scored very small distance error for large sets of top(K) while $SeeDB_Timelimit$ shows very low accuracy and huge distance error. As illustrated in 15c The execution costs of the proposed algorithms remains stable on different time limits and it shows the similar behavior figure 16c the overhead is fixed along different top-k views.

To sum up, the proposed algorithms improved the quality of results according two the evaluation metrics that used along different K sizes and various times limits. Moreover, the algorithms overhead is comparatively small with the total execution of SeeDB baseline.

5 Related Works

Interactive Data Visualization Tools: have interested the research community over the past few years, and it has presented a number of interactive data analytics tools such as ShowMe, Polaris, and Tableau [15,21,16,6]. Similar visualization specification tools have also been introduced by the database community, including Fusion Tables [9] and the Devise [19] toolkit. Unlike SeeDB, which recommends visualizations automatically by exploring the entire views space, these tools place the onus on the analyst to specify the visualization to be generated. For datasets with a large number of attributes, it is unfeasible for the analyst to manually study all the attributes; hence, interactive visualization needs to be augmented with automated visualization techniques.

A few recent systems have attempted to automate some aspects of data analysis and visualization. Profiler is one such automated tool that allows analysts to detect anomalies in data [15]. Another related tool is VizDeck [16], which, given a dataset, depicts all possible 2-D visualizations on a dashboard that the user can control by reordering or pinning visualizations. Given that VizDeck generates all visualizations, it is only meant for small datasets; additionally, the VizDeck does not discuss techniques to speed-up the generation of these visualizations.

Statistical analysis and graphing packages such as R, SAS and Matlab could also be used generate visualizations, but they lack the ability to filter and recommend 'interesting' visualizations.

OLAP: there has been some work on browsing data cubes, allowing analysts to variously find *explanations* for why two cube values were different, to find which neighboring cubes have similar properties to the cube under consideration, or get suggestions on what unexplored data cubes should be looked at next [13,24,25].

Database Visualization Work: Fusion tables [9] allow users to create visualizations layered on top of web databases; they do not consider the problem of automatic visualization generation. Devise [10] translated user-manipulated visualizations into database queries.

Although the aforementioned approaches provide assistance in query visualization, they lack the ability to automatically recommend interesting visualizations, except SeeDB which provides different optimization techniques to automatically recommend interesting visualizations while avoiding unnecessary visualizations by utilizing two kinds of optimization techniques as explained next.

Visualizations Pruning in SeeDB: SeeDB implement an execution engine to reduce latency in assessing the collection of aggregate views which it applies two kinds of optimizations: sharing, where aggregate view queries are combined to share computation as much as possible, and pruning, where aggregate view queries corresponding to low utility visualizations are dropped from consideration without scanning the whole dataset. SeeDB developed a phased execution framework,each phase operates on a subset of the dataset. Phase i of n operates on the i^{th} of n equally-sized partitions of the dataset. The execution engine begins with the entire set of aggregate views as follows: During phase i,the SeeDB [30] modifies partial results for the views still under consideration using the i^{th} fraction of the dataset. The execution engine applies sharing-based optimizations to minimize scans on this i^{th} fraction of the dataset. At the end of phase i,the execution engine uses pruning-based optimizations to determine which aggregate views to discard. The partial results of each aggregate view on the fractions from 1 through i are used to estimate the quality of each view, and the views with low utility are discarded.

The execution engine uses pruning optimizations to determine which aggregate views to discard. Specifically, partial results for each view based on the data processed so far are used to estimate utility and views with low utility are discarded. SeeDB execution engine supports two pruning schemes. The first uses confidence-interval techniques to bound utilities of views, while the second uses multi-armed bandit allocation strategies to find top utility views.

- Confidence Interval-Based Pruning: The first pruning scheme uses worst-case statistical confidence intervals to bound views utilities. This technique is similar to top-k based pruning algorithms developed in other contexts [26]. It works as follows: during each phase, it keeps an estimate of the mean utility for every aggregate view V_i and a confidence interval around that mean. At the end of a phase, it applies the following rule to prune low-utility views: If the upper bound of the utility of view V_i is less than the lower bound of the utility of k or more views, then V_i is discarded.
- Multi-Armed Bandit Pruning: Second pruning scheme employs a Multi-Armed Bandit strategy (MAB) [30, 2]. In MAB, an online algorithm repeatedly chooses from a set of alternatives over a sequence of trials to maximize reward. This variation is identical to the problem addressed by SeeDB: the goal is find the visualizations (arms) with the highest utility (reward). Specifically, SeeDB adapts the Successive Accepts and Rejects algorithm from [2] to find arms with the highest mean reward. At the end of every phase, views that are still under consideration are ranked in order of their utility means. We then compute two differences between the utility means: $\Delta 1$is the difference between the highest mean and the $k+1^{st}$ highest mean, and Δn is the difference between the lowest mean and the k^{th} highest mean. If $\Delta 1$ is greater than Δn, the view with the highest mean is accepted as being part of the top-k (and it no longer participates in pruning computations). On the other hand, if Δn is higher, the view

with the lowest mean is discarded from the set of views in the running. [6] proves that under certain assumptions about reward distributions, the above technique identifies the top-k arms with high probability.

However, SeeDB pruning schemes experience some limitations, as they assume fixed data distribution [30,29] for sampling to estimate the utility of views and require large samples for pruning low utility views with high guarantees. Moreover, aggregate functions MAX and MIN are not docile to sampling-based optimizations.

Offline visualizations in SeeDB: SeeDB prunes redundant views [30] : (1) For each table, it first determines the entire space of aggregate views. (2) Next, it prunes all aggregate views containing attributes with 0 or low variance since corresponding visualizations are unlikely to be interesting. (3) For each remaining view V_i, SeeDB computes the distribution for reference views on the entire dataset . (4) The resulting distributions are then clustered based on pairwise correlation. (5) From each cluster, SeeDB selects one view to compute as a cluster representative and store stubs of clustered views for sub- sequent use. At run time, the view generator accesses previously generated view stubs, removes redundant views and passes the remaining stubs to the execution engine.

6 Conclusion

Finding top interesting visualizations by exploring a specified number of visualizations or a limited execution time budget, while persevering the quality and the accuracy of the recommended views is challenging and emerging problem. In this paper, we propose an efficient framework Realtime Scoring Engine that assists the analyst to limit the exploration of visualizations for a specified number of visualizations or certain execution time quote to recommend a set of views that meets the analysts budgets. The implementation of RtSEngine incorporates our proposed approaches to priotrize attributes that form all possible visualizations in dataset based on their statistical proprieties such as selectivity ratio, data distribution, and number of distinct values then recommends the views created from top attributes. In addition, we present visualizations cost-aware techniques that estimate the retrieval and computation costs of all visualizations then utilise the estimated costs with to recommend views while considering their costs. Furthermore, our comparative experimental study assess the quality of visualizations and the overhead obtained by applying theses techniques on both synthetic and real datasets. The experiments show the effectiveness and efficiency of proposed approaches to recommend potential visualizations running on different time and space limits.

References

1. Barbará, D., DuMouchel, W., Faloutsos, C., Haas, P.J., Hellerstein, J.M., Ioannidis, Y.E., Jagadish, H.V., Johnson, T., Ng, R.T., Poosala, V., Ross, K.A., Sevcik, K.C.: The new jersey data reduction report. IEEE Data Eng. Bull. **20**(4), 3–45 (1997)
2. Bubeck, S., Wang, T., Viswanathan, N.: Multiple identifications in multi-armed bandits. In: Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013, pp. 258–265 (2013). URL `http://jmlr.org/proceedings/papers/v28/bubeck13.html`
3. Charikar, M., Chaudhuri, S., Motwani, R., Narasayya, V.R.: Towards estimation error guarantees for distinct values. In: Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, May 15-17, 2000, Dallas, Texas, USA, pp. 268–279 (2000)
4. Chaudhuri, S.: An overview of query optimization in relational systems. In: Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS '98, pp. 34–43. ACM, New York, NY, USA (1998). DOI 10.1145/275487.275492. URL `http://doi.acm.org/10.1145/275487.275492`
5. Chaudhuri, S., Motwani, R., Narasayya, V.R.: Random sampling for histogram construction: How much is enough? In: SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA., pp. 436–447 (1998)
6. Fisher, D.: Hotmap: Looking at geographic attention. IEEE Trans. Vis. Comput. Graph. **13**(6), 1184–1191 (2007)
7. Getoor, L., Taskar, B., Koller, D.: Selectivity estimation using probabilistic models. In: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, SIGMOD '01, pp. 461–472. ACM, New York, NY, USA (2001). DOI 10.1145/375663.375727. URL `http://doi.acm.org/10.1145/375663.375727`
8. Gilbert, A.C., Kotidis, Y., Muthukrishnan, S., Strauss, M.: Optimal and approximate computation of summary statistics for range aggregates. In: Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 21-23, 2001, Santa Barbara, California, USA (2001)
9. Gonzalez, H., Halevy, A.Y., Jensen, C.S., Langen, A., Madhavan, J., Shapley, R., Shen, W., Goldberg-Kidon, J.: Google fusion tables: web-centered data management and collaboration. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2010, Indianapolis, Indiana, USA, June 6-10, 2010, pp. 1061–1066 (2010)
10. Hellerstein, J.M., Haas, P.J., Wang, H.J.: Online aggregation. In: SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA., pp. 171–182 (1997)
11. Hou, W.C., Ozsoyoglu, G.: Statistical estimators for aggregate relational algebra queries. ACM Trans. Database Syst. **16**(4), 600–654 (1991). DOI 10.1145/115302.115300. URL `http://doi.acm.org/10.1145/115302.115300`
12. Ioannidis, Y.: The history of histograms (abridged). In: Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29, VLDB '03, pp. 19–30. VLDB Endowment (2003). URL `http://dl.acm.org/citation.cfm?id=1315451.1315455`
13. Jagadish, H.V.: Review - explaining differences in multidimensional aggregates. ACM SIGMOD Digital Review **1** (1999)
14. Jang, M.H., Kim, S.W., Faloutsos, C., Park, S.: A linear-time approximation of the earth mover's distance. In: Proceedings of the 20th ACM International Conference on Information and Knowledge Management, CIKM '11, pp. 505–514. ACM, New York, NY, USA (2011). DOI 10.1145/2063576.2063652. URL `http://doi.acm.org/10.1145/2063576.2063652`
15. Kandel, S., Parikh, R., Paepcke, A., Hellerstein, J.M., Heer, J.: Profiler: Integrated statistical analysis and visualization for data quality assessment. In: Proceedings of the International Working Conference on Advanced Visual Interfaces, pp. 547–554. ACM (2012)
16. Key, A., Howe, B., Perry, D., Aragon, C.R.: Vizdeck: self-organizing dashboards for visual analytics. In: Proceedings of the ACM SIGMOD International Conference on

Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012, pp. 681–684 (2012)
17. Lahdenmaki, T., Leach, M.: Relational Database Index Design and the Optimizers. John Wiley & Sons (2005)
18. Lipton, R.J., Naughton, J.F., Schneider, D.A.: Practical selectivity estimation through adaptive sampling. In: Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, SIGMOD '90, pp. 1–11. ACM, New York, NY, USA (1990). DOI 10.1145/93597.93611. URL http://doi.acm.org/10.1145/93597.93611
19. Livny, M., Ramakrishnan, R., Beyer, K.S., Chen, G., Donjerkovic, D., Lawande, S., Myllymaki, J., Wenger, R.K.: Devise: Integrated querying and visualization of large datasets. In: SIGMOD 1997, Proceedings ACM SIGMOD International Conference on Management of Data, May 13-15, 1997, Tucson, Arizona, USA., pp. 301–312 (1997)
20. Mackert, L.F., Lohman, G.M.: R* optimizer validation and performance evaluation for local queries. SIGMOD Rec. **15**(2), 84–95 (1986). DOI 10.1145/16856.16863. URL http://doi.acm.org/10.1145/16856.16863
21. Mackinlay, J.D., Hanrahan, P., Stolte, C.: Show me: Automatic presentation for visual analysis. IEEE Trans. Vis. Comput. Graph. **13**(6), 1137–1144 (2007)
22. Mannino, M.V., Chu, P., Sager, T.: Statistical profile estimation in database systems. ACM Comput. Surv. **20**(3), 191–221 (1988)
23. Piatetsky-Shapiro, G., Connell, C.: Accurate estimation of the number of tuples satisfying a condition. In: SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984, pp. 256–276 (1984)
24. Sarawagi, S.: User-adaptive exploration of multidimensional data. In: VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt, pp. 307–316 (2000)
25. Sathe, G., Sarawagi, S.: Intelligent rollups in multidimensional OLAP data. In: VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases, September 11-14, 2001, Roma, Italy, pp. 531–540 (2001)
26. Serfling, R.J.: Probability inequalities for the sum in sampling without replacement. The Annals of Statistics pp. 39–48 (1974)
27. Stillger, M., Lohman, G.M., Markl, V., Kandil, M.: Leo - db2's learning optimizer. In: Proceedings of the 27th International Conference on Very Large Data Bases, VLDB '01, pp. 19–28. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001). URL http://dl.acm.org/citation.cfm?id=645927.672349
28. Stolte, C., Hanrahan, P.: Polaris: A system for query, analysis and visualization of multi-dimensional relational databases. In: Proceedings of the IEEE Symposium on Information Vizualization 2000, INFOVIS '00, pp. 5–. IEEE Computer Society, Washington, DC, USA (2000). URL http://dl.acm.org/citation.cfm?id=857190.857686
29. Vartak, M., Madden, S., Parameswaran, A., Polyzotis, N.: Seedb: towards automatic query result visualizations. Tech. rep., Technical Report, data-people. cs. illinois. edu/seedb-tr. pdf
30. Vartak, M., Madden, S., Parameswaran, A.G., Polyzotis, N.: SEEDB: automatically generating query visualizations. PVLDB **7**(13), 1581–1584 (2014)