SEEDB: A Visualization Recommendation System

ABSTRACT

Data analysts operating on large volumes of data often rely on visualizations to interpret the results of queries. However, finding the right visualization for a query is a laborious and time-consuming task. We demonstrate SEEDB, a system that partially automates the task of finding the right visualization for a query: given a query, SEEDB explores the space of all possible visualizations, and automatically identifies and then recommends to the data scientist those visualizations it finds to be the most "interesting" or "useful". In our demonstration, conference attendees will see SEEDB in action for multiple queries on multiple real-world datasets.

1. INTRODUCTION

Data analysts must sift through very large volumes of data to identify trends, insights, or anomalies. Given the scale of data, and the relative ease and intuitiveness of examining data visually, analysts often use visualizations as a tool to identify these trends, insights, and anomalies. However, selecting the "right" visualization often remains a laborious and time-consuming task.

We illustrate this using an example. Consider a dataset containing sales records for a nation-wide chain of stores. Let's say the store's data analyst is interested in examining how a newly-introduced range of electronic heating devices, "Laserwave Oven", has been doing over the past year. The results of their analysis will inform business decisions for the chain, including marketing strategies, as well as the possiblity of introduction of a new heating device, the "Saberwave Oven".

Consider the following typical data analysis workflow that the analyst may use: (1) The analyst poses a query to select the subset of data that they are interested in exploring. For instance, for the example above, they may issue the query:

```
Q = SELECT * FROM Sales WHERE Product = "Laserwave"
```

Notice that the results for this query may have (say) several million records each with several hundred attributes. Thus, directly perusing the query result is simply infeasible. (2) Next, the analyst considers various ways of visualizing or viewing the results of this query. To do this, the analyst will use operations like binning, grouping, and aggregations, and then generate visualizations such as scatterplots, histograms, or bar charts. In our example, the analyst may generate and examine the visualizations or views corresponding to total sales by store, quantity in stock by region, or average profits by month. To generate, say, average profits by month, the analyst would need to bin the sale dates into months, group the sales records by month, and then compute the average profits by month. This operation can be expressed as the following query:

```
Q' = SELECT AVG(profit) FROM Sales WHERE
Product = "Laserwave" GROUP BY month(saledate)
```

The result of the above query is a two-column table that can then be visualized as a bar-chart. Table 1 and Figure 1 respectively show an example of the results of this view and the associated visualization. The analyst generates all possible views or visualizations of the query result of the form described above. (3) The analyst then manually examines each view or visualization and decides which views are "interesting". This is the critical and time-consuming step. Naturally, what makes a view interesting depends on the application semantics and the trend we are comparing against. For instance, the view of Laserwave sales by month shown in Figure 1 may be interesting if the overall sales of products shows the opposite trend (e.g. Figure 2). However, the same view may be uninteresting if the sales of all products follows the same trend (Figure 3). Thus, we posit that a view is *potentially "interesting" if it shows a trend in the subset of data that the analyst is interested in (i.e., Laserwave product-related data) that deviates from the equivalent trend in the overall dataset*. Of course, the analyst must decide if this deviation is truly an insight for this application. (4) Once the analyst has identified interesting views, the analyst may then either share these views with others, may further interact with the displayed views or visualizations by drilling down or rolling up, or may start afresh with a new query.

Of the four steps in the workflow described above, the ones that are especially repetitive and tedious are steps (2) and (3), where the analyst generates all possible views, and examines each of them. The goal of our system, SEEDB, is to partially automate these two steps. Given a query Q indicating the subset of data that the analyst is interested in (i.e., the result of step (1)), SEEDB automatically *identifies and highlights to the analyst the most potentially interesting views using automated mechanisms based on deviation*. That is, we measure how much these views deviate from the corresponding views on the underlying dataset (just like Figure 1 vs. Figures 2 or 3.) By doing so, we eliminate the laborious process of stepping through all possible views that the analyst currently performs. Once we recommend potentially interesting views, the analyst can limit their analysis to these views, and can further make the final deci-

Proceedings of the VLDB Endowment, Vol. 7, No. 4

Table 1: Average Profit by Month for Laserwave

Month	Profit($)
Jan	180.55
Feb	145.50
March	122.00
April	90.13

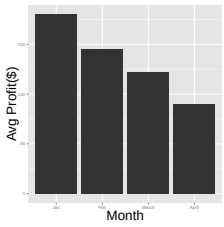

Figure 1: Visualization of average profit by month for Laserwave

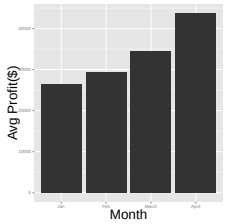

Figure 2: Average profit by month for all products — Scenario A

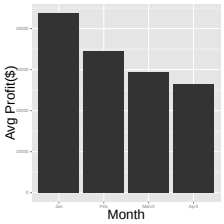

Figure 3: Average profit by month for all products — Scenario B

sion as to whether the recommended views are indeed interesting given the semantics of the application.

We described our vision for SEEDB, along with the associated research challenges in a companion vision paper [8]. In this demonstration proposal, we present our first SEEDB prototype, addressing some of the challenges that were listed in that vision paper. In particular, SEEDB is built as a "wrapper" that can be overlaid on any existing traditional database system, and given a query, generates and recommends interesting visualizations. Note that even architecting SEEDB as a wrapper requires us to address the following challenges: (a) we need to determine metrics that accurately measures the "deviation" of a view with respect to the equivalent view on the entire database (e.g., Figure 1 vs. 2), while at the same time ensure that SEEDB is not tied to any particular metric(s); (b) the number of potential views (or visualizations) increases as the square of the number of attributes in a table (we will demonstrate this in subsequent sections) — so generating and evaluating all views even for a moderately sized dataset (e.g. 1M rows, 100 attributes) can be prohibitively expensive; as a result, SEEDB must intelligently explore this space; (c) computing each view and its "deviation" from the same view on the underlying dataset independently is expensive and wasteful, and hence SEEDB must share computation as much as possible; and (d) since visual analysis must happen in real-time, SEEDB may have to trade-off accuracy of visualizations or accuracy of estimation of "interestingness" for reduced latency. In Section 3, we describe how SEEDB addresses these challenges.

Related Work: Over the past few years, the research community has introduced a number of interactive data analytics tools: tools such as ShowMe, Polaris, and Tableau [12, 7], as well as allow analysts to specify and generate visualizations, while tools like Profiler allow analysts to detect anomalies in data. Unlike SEEDB, which recommends visualizations, here the onus is on the analyst to specify the visualization to be generated. Similar visualization specification tools have also been introduced by the database community, including Fusion Tables [5] and the Devise [6] toolkit.

There has been some work on browsing data cubes, allowing analysts to find explanations, get suggestions for next cubes to visit, or identify generalizations or patterns starting from a single cube [9, 11, 10]. While we may be able to reuse the metrics from that line of work, the same techniques will not apply to visualizations.

2. PROBLEM STATEMENT

Given a database D and a query Q, SEEDB considers a number of views that can be generated from Q by adding relational operators, as we will see below. For the purposes of this discussion, we will refer to views and visualizations interchangeably, since it is straightforward to translate views into visualizations automatically: for example, there are straightforward rules that dictate how the view Table 1 must be transformed to give a visualization like Figure 1. Also for this discussion, we will limit the set of potential views to be those that generate a two column result, much like Table 1; however, SEEDB can generate and recommend visualizations corresponding to multiple (greater than two) column views. Lastly, for simplicity, we ignore *binning*: that is, given a view to be visualized, there are many ways of binning values to give the view. For instance, if we have average profits per day, we can bin the days into months, into weeks, or into years.

We consider a database D with a snowflake schema, with dimension attributes A, measure attributes M, and potential aggregate functions F over the measure attributes. We limit the class of queries Q posed over D to be those that select a horizontal fragment of the fact table: this selection can be done using selection predicates on the fact table, or on dimension tables via key-foreign-key joins. Overall, this class of queries allows the analyst to express their interest in examining facts (i.e., a slice of the dataset) that satisfy specific conditions. We denote the result of $Q(D)$ as D_Q.

Given such a query, SEEDB considers all views V_i that perform a group-by and some aggregation on D_Q. Since our resulting view must have two columns (though, as mentioned earlier, SEEDB does generate and recommend views with more than two columns), V_i can be represented as a triple (a, m, f), where $m \in M, a \in A, f \in F$, i.e., the view performs a group-by on a and applies the aggregation function f on a measure attribute m. Thus, $V_i(D_Q)$ can be expressed as the following SQL query:

$$\texttt{SELECT}\ a, f(m)\ \texttt{FROM}\ D_Q\ \texttt{GROUP BY}\ a$$

We call this *target view*.

As discussed in the previous section, SEEDB evaluates whether a view V_i is interesting by computing the deviation between the view applied to the selected data (i.e., D_Q) and the view applied to the entire database. The equivalent view on the entire database $V_i(D)$ can be expressed as:

$$\texttt{SELECT}\ a, f(m)\ \texttt{FROM}\ D\ \texttt{GROUP BY}\ a$$

We call this the *comparison view*.

Note that both these views, the target view and the comparison view give rise to a result containing two columns: a and $f(m)$. A two-column table can be converted into a probability distribution, where we normalize the values of $f(m)$ such that they sum to 1 over the various values of a. For our example in Table 1, the probability distribution of $V_i(D_Q)$, denoted as $P[V_i(D_Q)]$, is: (Jan: 180.55/538.18, Feb: 145.50/538.18, March: 122.00/538.18, April: 90.13/538.18) A similar probability distribution can be derived for $P[V_i(D)]$.

Then, the utility U of a view V_i is defined as the distance between the probability distribution of the target view and the probability distribution of the comparison view; formally:

$$U(V_i) = S(P[V_i(D_Q)], P[V_i(D)])$$

The utility of a view is our measure for whether the target view is "potentially interesting" as compared to comparison view: the

higher the value, the more the deviation from the comparison view, and the more likely the view is to be interesting to the analyst. Computing distance between probability distributions has been well studied in the literature, and SEEDB supports a variety of metrics to compute distance, including:

- **Earth Movers Distance (EMD)** [16]: Commonly used to measure differences between color histograms from images, EMD is a popular metric for comparing discrete distributions.
- **Euclidean Distance**: The L2 norm or Euclidean distance considers the two distributions are points in a high dimensional space and measures the distance between them.
- **Kullback-Leibler Divergence** [15]: K-L divergence measures the information lost when one probability distribution is used to approximate another.
- **Jenson-Shannon Distance** [14, 13]: Based on the K-L divergence, this distance measures the similarity between two probability distributions.

We are now ready to state the goals of SEEDB formally:

GOAL 2.1. *Given an analyst-specified query Q on a database D, a distance function S, and a positive integer k, find k views $V \equiv (a, m, f)$ that have the largest values of $U(V)$ among all the views that can be represented using a triple (a, m, f), while minimizing total computation time.*

Thus, SEEDB aims to find the k views (obtained by adding a single aggregate and group-by operator) that have the largest utility based on the function U.

3. SYSTEM ARCHITECURE

3.1 SEEDB overiew

Figure 4 shows the architecture of our system. Currently, SEEDB is implemeted as a layer on top of a database. While optimization opportunities are restricted by virtue of being outside the DBMS, it permits SEEDB to be used with a variety of existing databases.

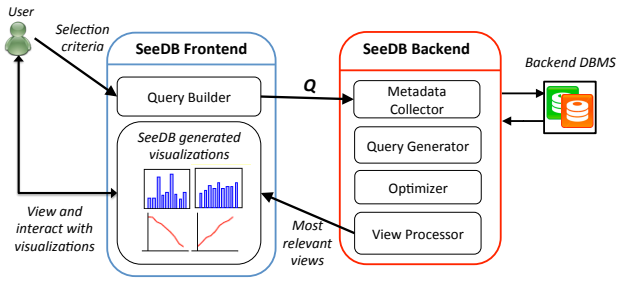

Figure 4: SeeDB Architecture

The SEEDB front end allows the user to issue queries through several means including raw SQL, via a query-builder and via preformulated queries (more in Section 3.2). Once user queries are sent to the backend, the Metadata Collector queries the relevant tables for information such as table sizes, column types, table statistics etc. This metadata is essential for the subsequent processing performed by SEEDB . The user queries along with metadata are then sent to the Query Generator. The purpose of the Query Generator is two-fold: first, it uses the metadata and user query to knock down the search space of possible view queries, and second, it generates the remaining view queries that must be run against the databse. The Optimizer is responsible for determining the best ways to combine view queries so that total query execution time is minimized. We discuss some of the optimizations performed by the Query Generator and Optimizer in Section 3.3. With the optimized queries in hand, SEEDB asks the DBMS to execute queries and return result to the View Processor that ingests combined query results in a streaming fashion and produces results for individual queries. The View Processor is also responsible for selecting the most interesting views and returning them to the user.

3.2 SeeDB Frontend

The SEEDB frontend has a two-fold purpose: allow the user to input a query, and to visualize and analyze the resulting views. We provide the user with three options for specifying an input query: (a) as raw SQL, (b) through a query builder that can allow users unfamiliar with SQL to formulate queries through an easy-to-use form-based interface, (c) through pre-defined query templates, e.g., queries that select outliers in a particular column. These templates are particularly useful since users are interested in anomalous data points.

Once a user submits a query to SEEDB the SEEDB engine evaluates various views and sends the most promising ones to the frontend. The frontend then determines the best ways to visualize these views (e.g. depending on data types being represented and number of distinct values) and displays the visualizations. The user can then examine these diverse views at a glance, explore specific views in detail and view metadata for each view (e.g. size of result, sample data, value with maximum change and statistics). The user can also slice-and-dice views further by selecting particular values of grouping attributes to explore. The user does this simply by selecting the relevant attribute values in the view. This automatically modifies the selection query and displays views for the subset of data selected. The user can of course revert back to the original views and continue exploring the data.

3.3 SeeDB Backend

One of the chief challenges in SEEDB is producing the most interesting views of the data in the minimum time. For this, SEEDB must perform optimizations at two stages: first, using prior knowledge such as statistics to prune out uninteresting views without examining table data; and second, minimizing the execution time for queries that are issued to the database. We first describe the basic SEEDB framework and then briefly discuss our optimizations.

3.3.1 Basic Framework

Given a user query Q, the basic version of SEEDB computes all possible view queries by adding a single aggregate and a single group-by clause to Q. Each of these view queries is executed independently at the backend along with an equivalent aggregate+group-by query on the complete underlying dataset. The two resulting distributions are compared using the chosen distance metric (Section 2) and the top k views with the largest utility are chosen. The frontend then visualizes the results. The basic approach is clearly inefficient and we can significantly improve performance.

3.3.2 View Space Pruning

To minimize the SEEDB response time, we aggressively prune view queries that are unlikely to generate interesting views. This pruning is based on prior knowledge about the data as well as access patterns for the table (if available). Specifically, we use variance estimates for table columns and correlation measures between columns of the table. If two columns in the table have high correlation, the views generated by grouping with respect to these two

columns will be very similar. We therefore only generate a single view for the pair of columns. We can similarly use statistical properties of measure attributes to prune views further. Finally, if access patterns for the table are available, we can use them to prune views with attributes that are rarely accessed and are therefore likely to be unimportant.

3.3.3 View Query Optimizations

Since SEEDB executes many similar queries, we clearly can do better than the basic framework presented above, and SEEDB implements various such optimizations. Our optimization strategies include:

1. *Rewrite view query*: Since similar group-by and aggregate queries are executed on the results of user query Q and the underlying dataset, one straighforward optimization is to combine these two queries into one. XXX actual queries.

2. *Single Group-by Multiple Aggregates*: A large number of view queries have the same group-by clause but aggregates over different attributes. Therefore, SEEDB combines all view queries with the same group-by clause into a single view query. This rewriting provides a speed up linear in the number of aggregate attributes.

3. *Multiple Aggregate Computation*: SEEDB computes a large number of group-bys. One optimization is to combine queries with different group-by attributes into a single query with multiple group-bys attributes. For instance, consider view queries $V(Q, A_1, G_1)$, $V(Q, A_1, G_2)$ $\ldots V(Q, A_1, G_n)$. Instead of executing them individually, we can rewrite them into a single view query $V(Q, A_1, (G_1, G_2 \ldots G_n))$. While this optimization can reduce the number of queries executed, the number of group by attributes we can include in a single query depends on the correlation between values of the various attributes (this affects number of distinct groups) and the working memory. We can compute the optimal combinations of group-bys by modeling the problem as a variant of bin-packing.

4. *Sampling*: The optimization that can have the most impact in terms of efficiency is to reduce the number of tuples examined by constructing a data sample and running all queries against the sample. As expected, the sampling technique and size of the sample can affect the accuracy of the generated views.

4. DEMO WALKTHROUGH

We propose to demonstrate the functionality of SEEDB through hands-on interaction with a variety of datasets. Our goals are two fold: (1) demonstrate the utility of a tool like SEEDB in surfacing interesting trends for a query and (2) demonstrate that we can return high quality results efficiently for datasets with varying sizes and number of attributes.

Demonstrating Utility: To show the utility of SEEDB in a real-world scenario, we will provide conference attendees three diverse datasets that they can explore and interact with. Attendees can pose ad-hoc or pre-selected queries on various datasets and evaluate the visualizations returned. The evaluation is based on whether the visualizations surface "interesting" aspects of the queried data and whether the right visualizations have been selected. To aid the evaluation of visualizations, the demo verion of SEEDB will have the option of showing "bad" visualizations too, i.e. visualizations that were predicted to have low utility. The purpose behind providing some pre-selected queries (and interesting information about their results) is to allow attendees to confirm that the tool does indeed reproduce known information about these queries. The attendees will also have the option of trying various utility metrics as described in Section 2. The demo datasets will include:

1. **Store Orders dataset** [4]: This dataset is often used by Tableau [3] as a canonical dataset for business intelligence applications. It consists of information about orders placed in a store including products, prices, ship dates, geographical information, profits, and so on. This dataset is well-studied by users learning how to use Tableau, with several web-pages dedicated to discovering interesting trends hidden in this dataset [**?**]. Attendees using SEEDB will be able to identify very quickly the same insights and trends that Tableau users have discovered over many years. This dataset will also enable us to demonstrate how SEEDBcan correctly deal with numeric, categorical, and geographic data.

2. **Election Contribution dataset** [1]: This dataset is an example of a real-world dataset that is typically analyzed by non-expert data analysts, such as journalists or historians. This dataset will enable us to demonstrate to the attendees how non-experts can quickly arrive at interesting visualizations via the intuitive user interface.

3. **Medical dataset [2]:** This dataset is an example of a real-world dataset that a researcher (here, a clinical researcher) might use over the course of his/her work. This data has a schema that is more complex than the the election or store one, and is of larger size too.

Demonstrating Speed and Optimizations: This demonstration scenario will use an enhanced user interface and synthetic datasets with varying sizes, attribute numbers and attribute distributions. Attendees can examine the efficiency of SEEDB by tuning various "knobs" such as data size, number of attributes, optimizations applied etc.

Thus, through our demonstration of SEEDB we seek to illustrate that (a) it is possible to automate labor-intensive parts of data analysis, (b) aggregate based views are a powerful means of identifying interesting trends in data, and (c) we can use various optimizations to enable data analysis in real-time for a range od datasets.

5. REFERENCES

[1] Fec presidential campaign finance. [Online; accessed 3-March-2014].
[2] Mimic ii database. [Online; accessed 3-March-2014].
[3] Tableau public. [Online; accessed 3-March-2014].
[4] Tableau superstore data. [Online; accessed 3-March-2014].
[5] H. Gonzalez et al. Google fusion tables: web-centered data management and collaboration. In *SIGMOD Conference*, pages 1061–1066, 2010.
[6] M. Livny et al. Devise: Integrated querying and visualization of large datasets. In *SIGMOD Conference*, pages 301–312, 1997.
[7] J. D. Mackinlay et al. Show me: Automatic presentation for visual analysis. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1137–1144, 2007.
[8] A. Parameswaran, N. Polyzotis, and H. Garcia-Molina. Seedb: Visualizing database queries efficiently. In *VLDB*, volume 7, pages 325–328, 2013.
[9] S. Sarawagi. Explaining differences in multidimensional aggregates. In *VLDB*, pages 42–53, 1999.
[10] S. Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, pages 307–316, 2000.
[11] G. Sathe and S. Sarawagi. Intelligent rollups in multidimensional olap data. In *VLDB*, pages 531–540, 2001.
[12] C. Stolte et al. Polaris: a system for query, analysis, and visualization of multidimensional databases. *Commun. ACM*, 51(11):75–84, 2008.
[13] C. Wang and H.-W. Shen. Information theory in scientific visualization. *Entropy*, 13(1):254–273, 2011.

[14] Wikipedia. Jensen shannon divergence — wikipedia, the free encyclopedia, 2013. [Online; accessed 16-July-2013].

[15] Wikipedia. Kullback leibler divergence — wikipedia, the free encyclopedia, 2013. [Online; accessed 16-July-2013].

[16] Wikipedia. Statistical distance — wikipedia, the free encyclopedia, 2013. [Online; accessed 16-July-2013].