Paper reading summary

Shu SHANG

Paper: SEEDB: Efficient Data-Driven Visualization Recommendations to Support Visual Analytics
Authors: Manasi Vartak (MIT), Sajjadur Rahman (UIUC), Samuel Madden (MIT), Aditya Parameswaran (UIUC), Neoklis Polyzotis (Google)

Remarks

- Fisrt version: 20/04/2017
- Second version: 24/04/2017 (reviewed 20/04/2017 with Ioana)

Context

This paper mainly proposed an visualization recommendation engine called **SEEDB**, which is designed to facilitate data exploration, especially data with high dimension properties (high-dimensional datasets).

To design such recommendation engine, two problems arise: **scale** and **utility**. Scale means that the engine should be able to compute the recommendation within interactive time scale (for instance, engine with too much latency will give expert a bad experience). Utility means that the engine should be able to rank visualizations according to their likely intersts for the expert.

For scalability, the authors proposed **two optimization mechanisms** (1. sharing-based optimization 2. pruning-based optimization). For utility, the authors proposed a utility metric based on deviation from some reference data (Will be detailed after).

At last, the authors showed some experimental results and a user study to demonstrate the effectiveness of this method.

Problem

The Problem is: **How to help users (experts) discover the data efficiently and interactively?**

How to decide what is interesting?

When confronted with high-dimensional datasets, identifying the interestingness of a data visualization can be difficult because different dimensions and the combinations could explode the number of visualization that should be analyzed.

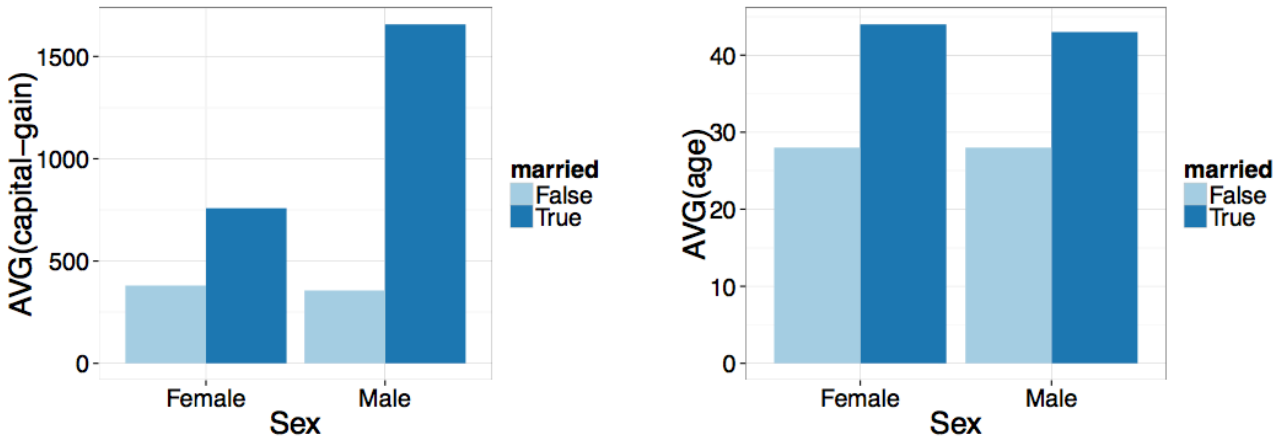
The interestingness of a visualization depends on plenty of factors, such as distribution, deviation, skewness... In this paper, a simple criterion based on deviation is proposed, a visualization is considered to be interesting if it shows a large deviation from some reference data (the detail will be shown in the **Solutions** part).

However, some challenges remain once we use a deviation utility metric:

- Firstly, even a dataset with a small number of attributes could generate a large set of visualizations, which means the number of visualizations grows very quickly with the number of attributes.
- Then, intuitively, evaluating each visualization based on this metric need to re-scan the underlying datasets over and over again, which is not optimal.
- As the tool should support exploratory analysis, low accuracy may be trade-off at some times to obtain an interactive speed.

Motivation example

The authors introduced an example of millennials. The researchers want to find some interesting insight from relations between different indicators like **marital status** and **average income**, or **marital status** and **average age**.



(a) Interesting Visualization *(b) Uninteresting Visualization*

The figure above is an example. The left figure is more interesting as we can see: although the difference between the average income for un-married male/female is not important, the difference of average income is huge for married male/female. However, the figure in the right-hand side is less interesting as it shows less deviation for male/female with different marital status.

NB: In this example, the **target dataset** (The dataset could share something interesting) is married adults and the **reference dataset** is unmarried adults. The deviation is thus defined between the target dataset and reference dataset.

The goal of this paper is to build a system that uses deviation as a mean to identify the most interesting

ones among a set of visualization candidates.

Solutions

Notations

D: database
F: the set of potential aggregate functions (count, avg...)
A: a set of dimension attributes that we can use to group by
M: a set of measure attribute that used to aggregate
Q: query Q, used to specify a subset of the dataset that an analyst wants to explore[1]
D_Q: the result of Q over D

NB:
[1] The class of query Q in this paper: select-project-join (SPJ) queries over the snowflake schema

SEEDB visualization

Each SEEDB visualization can be translated to **an aggregate over group-by query**.

Notations:

$V_i \leftrightarrow$ a visualization, represented by a triple (a, m, f), where $m \in M, a \in A, f \in F$.
we call this an aggregate view, or simply $view$.

$V_i(D)$: the results of grouping the data in D by a, and then aggregating the m values using f.

Definition of utility via deviation

We have two datasets D_Q (query dataset, target dataset) and D_R (reference dataset), given a view V_i, the deviation-based utility is computed as follows:

We apply the V_i to D_Q, get the target view:

$$Q_T = \text{SELECT } a, f(m) \text{ FROM } D_Q \text{ GROUP BY } a$$

Then we apply the V_i to D_R, get the reference view:

$$Q_R = \text{SELECT } a, f(m) \text{ FROM } D_R \text{ GROUP BY } a$$

The two queries above are called **view queries**.

To ensure the aggregate operations get the same scale, we **normalize** each summary into a probability distribution.

For instance, the probability distribution of the motivating example:

Average income vs. **sex**, the probability distribution for target view:

$$P[V_i(D)] \text{ is: (F: 0.52, M: 0.48)}$$

For reference view:

$$P[V_i(R)] \text{ is: (F: 0.31, M: 0.69)}$$

Average age vs. **sex**, the probability distribution for target view:

$$P[V_i(D)] \text{ is: (F: 0.5, M: 0.5)}$$

For the reference view:

$$P[V_i(R)] \text{ is: (F: 0.51, M: 0.49)}$$

Average income vs. **sex** and **Average age** vs **sex** are two different visualizations. We can qualitatively see that the distribution shows a larger diviation for the former one.

Formally, given an aggregate view V_i and probability distribution for the target view and reference view, the utility U of V_i is defined as the **distance between these two proba distributions**.

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

Examples of distance functions:

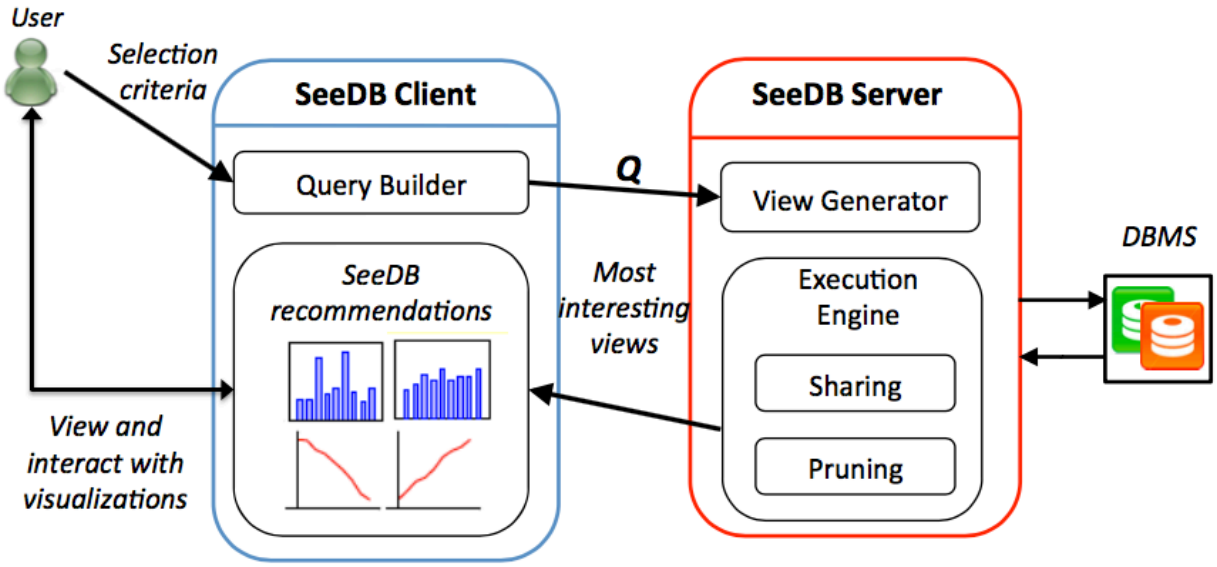
Earth Mover's Distance, Euclidean Distance, Kullback-Leibler Divergence, Jenson-Shannon Distance

SEEDB problem statement
Given a user-specified query Q on a database D, a reference dataset D_R, a utility function U as defined above, and a positive integer k, find k aggregate views $V \equiv (a, m, f)$ that have the largest values of $U(V)$ among all the views (a, m, f), while minimizing total computation time.

System design

The authors designed a system called SEEDB, consisting of:

- Front-end: support manually generating visualizations or obtaining data-driven recommendations, human in the loop.
- SEEDB can viewed as a middleware layer on top of any DBMS.
- SEEDB client is web-based, capturing user's input; SEEDB server produces output visualization, front-end renders the results.



SeeDB architecture

The figure above shows the architecture. The most interesting part is the **execution engine**. Two optimization techniques are used in this part: **sharing** and **pruning**. A phased execution framework is designed to integrate the optimization methods:

a phased execution framework

record1...1000 **Phase1**
record1001...2000 **Phase 2**
....
record9001...10000 **Phase 10**
...

(Next paragraph is taken directly from the paper)
Each phase operates on a subset of the dataset. Phase i of n operates on the ith of n equally-sized partitions of the dataset. The execution engine begins with the entire set of aggregate views under consideration. During phase i, SEEDB updates partial results for the views still under consideration using the ith fraction of the dataset. The execution engine applies sharing-based optimizations to minimize scans on this ith 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 for 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 retained aggregate views are then processed on the i + 1th round, and the process continues. **In this manner, the set of views under consideration decreases across these phases, with all aggregate views at the start of the first phase, and only the k views left at the end of the nth phase.**

Sharing-based Optimizations

Each visualization is translated into two queries. For a particular user input (Q and D_R), the queries

evaluated by the system are similar: they scan the underlying data, differ only in the attributes used for grouping and aggregation. This leads to opportunities for sharing computations.

Methods used:
1. Combine multiple Aggregates (same group-by, different aggregate)
2. Combine multiple GROUP BYs (pay attention to memory usage)
3. Combine target and reference view query
4. Parallel Query Execution

Pruning-Based Optimizations

In fact, most visualizations are low-utiliy, computing them is waste of resources. Two ways to avoid this:
1. Confidence Interval-Based Pruning (CI)
If the upper bound of the utility of view Vi is less than the lower bound of the utility of k or more views, then Vi is discarded.
2. Multi-Armed Bandit Pruning (MAB)
Our goal is to find the visualizations (arms) with the highest utility (reward)using **Successive Accepts and Rejects algorithm**.

Experimental results

Overall

- overall > 100X speedup
- 6-400X speedup from sharing
- 5X speedup from pruning without accuraray
- Gains improve on larger datasets

Without optimization techniques, SEEDB leads to latencies in the 100s of seconds for both ROW and COL: **latency increase linearly in the size of the dataset and number of views.**

Sharing optimizations

Combining Multiple Aggregates:

4X speedup for ROW and 3X for COL

Parallel Query Execution:

The optimal number of queries to run in parallel is approximately 16 (equal to the number of cores), suggesting that choosing a degree of parallelism equal to the number of cores is a reasonable policy

Combining Multiple Group-bys:

- 2.5X speedup in row stores

All sharing optimizations:

- 40X speedup for row stores and 6X speed up for columns sotres

Pruning optimizations

In pruning optimization part, besides latency (to measure if the optimization techniques work well), two additional metrics for evaluating the result (top-k recommened visualizations) proposed: **accuracy** and **utility distance**.

Distance function used: arth mover's distance.

Accuracy: $\{V_T\}$ the set of aggregate views with highest utility, $\{V_S\}$ the set of aggregate views returned by SEEDB. The accuracy is defined as follows:

$$\frac{1}{|\{V_i\}|} \times |\{V_T\} \cap \{V_S\}|$$

Utility distance: the difference between the average utility of $\{V_T\}$ and the average utility of $\{V_S\}$:

$$\frac{1}{n}(\sum_i U(V_{T,i}) - \sum(U_{S,i}))$$

- The accuracy of pruning is **inversely proportional** to the difference between the **k-th** highest utility and the **k+1-st** utility

The MAB and CI strategy both produce results with accuracy > 75% and near-zero utility distance for a variety of k values. MBA performs slightly better than CI when utility values are closely spaced.

Latency

Both strategies provide a reductiion in latency of 50% or more relative to no pruning technique. For smaller k, reductions can be even better, closer to 90% (this is especially usefual when we only want to explore the top one or top two views)