Efficient Data Slice Search for Exceptional View Detection

Yohei Mizuno, Yuya Sasaki, Makoto Onizuka
Graduate School of Information Science and Technology, Osaka University, Japan
{mizuno.yohei, sasaki, onizuka}@ist.osaka-u.ac.jp

ABSTRACT

Business analysts repeatedly execute OLAP queries consisting of aggregate/group-by operations until they find trends, insights, and/or outliers. One interesting analysis is to identify particular data slices (subset of original data) that generate exceptional views apart from the average view generated from the whole original data. However, since the search space for identifying such data slices is quite large, efficient techniques are indispensable for the data slice search. In this paper, we propose 1) a framework that automatically identifies data slices that generate exceptional views for OLAP queries, and 2) an efficient algorithm that optimizes the data slice search. The algorithm reduces the search space by employing confidence intervals and removes redundant query processing by applying multi-query optimization for evaluating queries over multiple data slices. The experiments validate that our algorithm improves the performance eight times faster than without the optimizations.

Keywords

Exploratory analysis, Probability theory, OLAP

1. INTRODUCTION

The need for effective analysis of business data is widely recognized and many data mining techniques have been developed, such as online analytical processing (OLAP), association rule mining, clustering, classification, graph mining and stream data mining [5]. In particular, OLAP is one of frequently used techniques that largely contributes to business data analysis and it is effectively used to find trends, insights, and/or outliers from data. In typical analytical processing, data analysts reveal hidden knowledge from data by setting up hypotheses and testing them through investigation of the data. However, a major issue of analytical processing is that it is not easy for data analysts to set up effective hypotheses, because it requires them to deeply understand the data itself so that they properly choose interesting analytical dimensions (expressed by OLAP queries) and/or data slices (subset of data) for detailed analysis. So, a continuous process of trial and error is very common for analytical processing.

The exploratory analysis is a promising research area for solving the issue above [2, 6, 12, 13]. Morton et al. [6] investigate systems that support data analysts. One challenging goal is to develop data recommenders that would automatically identify datasets outside of the system that could improve the quality of analysis result. Buoncristiano et al. [2] study a database exploration model assuming conversations between exploratory users and the system. The conversation goes on so that the system identifies queries that reveal significant deviation from uniform distribution. As a more concrete example, SEEDB [12, 13] automatically explores various types of OLAP queries in a multidimensional space and identifies query q that maximizes the deviation between $q(D)$ and $q(S)$, where D is the whole data and S is a given data slice. However, SEEDB is not applicable to an opposite case; to search for data slices that generate views that are deviated from the view generated from whole data with respect to a given OLAP query. An example of data slice search is to identify particular branch office (say, Kyoto office) or certain year (say, 2001) in sales databases such that its generated analysis result is deviated from the one generated from whole data. Moreover, since the search space of data slice search is huge, efficient techniques are indispensable for the data slice search.

To achieve efficient data slice search, we propose a framework that automatically identifies data slices that generate exceptional views from OLAP queries and an efficient algorithm that optimizes the data slice search. The algorithm reduces the search space by employing the confidence intervals [10] and removes redundant query processing by applying multi-query optimization techniques when searching for multiple data slices [9].

1.1 Running example

We give a running example of business analysis. In sales data analysis, it is important to identify particular products whose sales trends are deviated from the average sales of all products. Figure 1 shows normalized monthly sales of all products, products purchased by men, and clothes. We observe that the trend of the products purchased by men is quite close to that of all products, which is not interesting. In contrast, the trend of the clothes largely differs from that of all products. We can see that the sales of the clothes are affected by seasons particularly in March. This suggests that we should start a new discount service for clothes in April so as to keep the clothes sales high.

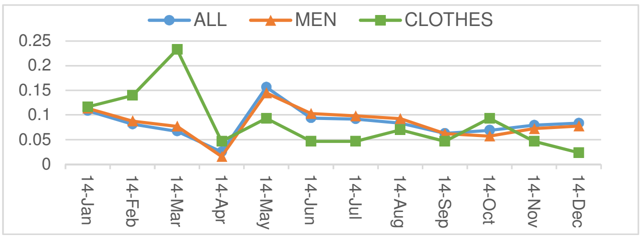

Figure 1: Normalized monthly sales of all products, products purchased by men, and clothes

This paper is organized as follows. We describe an overview of data slice search for exceptional view detection in Section 2 and confidence intervals derived from probability inequality in Section 3. We give the details of efficient algorithm for data slice search in Section 4. The performance evaluation is given in Section 5. We describe related work in Section 6 and conclude this paper in Section 7.

2. OVERVIEW OF DATA SLICE SEARCH

In this section, we describe a problem of identifying data slices that generate exceptional views for a given query. We give the problem definition of data slice search in Section 2.1, and then show its procedures in Section 2.2.

2.1 Problem definition

We focus on relational database D. D is a collection of records $X_1, X_2, \cdots, X_{|D|}$, where $|D|$ is the number of the records of D. X_i $(1 \leq i \leq |D|)$ is composed of measure attributes (say, prices or amounts) and dimension attributes B (say, product categories or store names). We denote value of measure attribute m of X_i as X_i^m. We define *data slice* S as a subset of D cuboid sliced by choosing a single value Y for $b \in B$ as follows:

$$S := \sigma_{b=Y}(D) \tag{1}$$

where σ is a select operation in relational algebra. We call this operation as *slicing query*. An OLAP query is given by data analysts in advance. We define a set of data slices $\mathbb{S}$ and OLAP query q as follows:

$$\mathbb{S} := \bigcup_{i=1}^{|B|} \{\sigma_{b_i=Y}(D) | Y \in values(b_i)\}, \tag{2}$$

$$q(S) := {}_gG_{a=f(m)}(S) \tag{3}$$

where $|B|$ is the number of dimension attributes, $values(b_i)$ is a set of unique values of b_i, g is a dimension attribute for group-by, m is a measure attribute for aggregate attribute, a is an aggregated value of m, and f is an aggregate function. f calculates either the number of the values of m (COUNT), the mean of the values of m (AVG), or the sum of the values of m (SUM). G groups the records using g and aggregates the grouped values of m using f. Those results are a sequence of objects consisting of the unique values of g and the aggregated values of m. The problem here is to identify the top-k data slices in $\mathbb{S}$ that generate the most extreme exceptional views (expressed as $q(S)$) for given query q, defined as follows:

Definition 1 $\underset{S \in \mathbb{S}}{\overset{k}{\operatorname{argmax}}}\ deviation(q(D), q(S))$
where deviation is a function to measure the deviation between the view for query q of D and of data slice S.

Without the loss of generality, we use Euclidean Distance for the deviation function[1].

2.2 Procedure

By using the running example in Section 1.1, we describe the procedure we automatically identify data slices that generate exceptional views.

1. To evaluate query q over D. In the running example, q is defined as a aggregate/group-by query for monthly sales as follows.
   $q(D) = \text{SELECT } month, \text{SUM}(price) \text{ FROM } D \text{ GROUP BY } month$
2. To obtain set of data slices $\mathbb{S}$ from D by choosing each single value $Y \in values(b_i)$ for each dimension $b_i \in B$. $S \in \mathbb{S}$ is generated by executing slicing queries, such as "gender = 'men'" and "category = 'clothes'".
3. To evaluate query q over all data slices $S \in \mathbb{S}$.
4. To compute the deviation between $q(D)$ and $q(S)$. For example, Euclidean Distance between the monthly sales of all products and those of men purchased products is computed.
5. To display the top-k deviated views to the users.

3. CONFIDENCE INTERVAL

We apply the confidence interval to efficiently identify top-k data slices when OLAP queries use aggregation functions. In this section, we first describe how we apply the confidence interval for mean value derived from the Hoeffding Serfling inequality [10].

The Hoeffding Serfling inequality is useful to derive the probability's upper bound without assuming the probability distribution of the population when the expected value for stochastic variable and its finite domain are given. From the Hoeffding Serfling inequality, the confidence interval for the mean value of aggregate attribute m of D (expressed as $\mu(D)$) can be derived. Let $\{x_1,...,x_{|x|}\}$ and $\{X_1,...,X_{|D|}\}$ be samples and population, and $|x|$ and $|D|$ be sample size and population size, respectively. The Hoeffding Serfling inequality is modified to have the following form.

$$Pr'(t) = Pr'[|\overline{x} - \mu(D)| \geq t] \tag{4}$$

where $\overline{x} = \frac{1}{|x|}\sum_{i=1}^{|x|} x_i^m$, $\mu(D) = \frac{1}{|D|}\sum_{i=1}^{|D|} X_i^m$. Since Eq.4 is derived from the Hoeffding Serfling inequality by replacing $\overline{x} - \mu(D)$ part with its absolute form, probability $Pr'(t)$ in Eq.4 has as twice as probability $Pr(t)$ in the Hoeffding Serfling inequality [10], where $Pr(t)$ is the probability that $\overline{x}$ is t bigger than $\mu(D)$. Therefore, suppose that $t > 0$, upper bound α can be inferred by using the following equation.

$$Pr'(t) \leq 2 \cdot \exp[\frac{-2|x|t^2}{(1-f_x)(max - min)^2}] = \alpha \tag{5}$$

where $f_x = \frac{|x|-1}{|D|}$, $min = \min\{X_i^m, i = 1, 2, \cdots, |D|\}$, $max = \max\{X_i^m, i = 1, 2, \cdots, |D|\}$. From Eq.4 and Eq.5, $(1-\alpha)$-confidence interval[2] for $\mu(D)$ can be derived as Eq.6.

$$\overline{x} - t < \mu(D) < \overline{x} + t \tag{6}$$

where $t = \sqrt{\frac{(1-f_x)(max-min)^2(\log 2 - \log \alpha)}{2|x|}}$.

[1]We can easily extend our technique to apply other deviation functions such as Manhattan Distance.
[2]We set α significance level at 0.05 in the experiments.

Furthermore, we express the range of the confidence interval in the following format.

$$range(\mu(D)) = [\overline{x} - t, \overline{x} + t]. \quad (7)$$

In the next place, we estimate the confidence interval for the sum of the values of m (expressed as $\eta(D)$). We can estimate this interval by using the following form obtained by multiplying $|D|$ to the terms in Eq.6.

Lemma 1 $|D|(\overline{x} - t) < \eta(D) < |D|(\overline{x} + t)$.

3.1 Confidence interval for size of data slices

We calculate aggregated values of m of data slices for the problem of identifying data slices that generate exceptional views for a given query. We need to estimate the aggregated values from samples so as to reduce the data slice search space. We derive expressions estimating confidence intervals for the aggregated values of m of the data slice $\sigma(D) \subseteq D$. Firstly, we derive the confidence interval for the size of $\sigma(D)$ (expressed as $|\sigma(D)|$). To estimate $|\sigma(D)|$ using Lemma 1, we define a function that judges whether an element of the samples x_i belongs to $\sigma(D)$ as follows:

$$exist(x_i) = \begin{cases} 1 & (x_i \in \sigma(D)) \\ 0 & (otherwise). \end{cases} \quad (8)$$

Furthermore, we define a ratio (expressed as $\overline{s}$) of the elements belongs to $\sigma(x)$ to the samples using *exist* as follows:

$$\overline{s} = \frac{|\sigma(x)|}{|x|} = \frac{1}{|x|}\sum_{i=1}^{|x|} exist(x_i). \quad (9)$$

By definition of *exist*, *min* in t equals zero, and *max* in t equals one in Eq.6. Therefore, we can estimate a ratio of $\sigma(D)$ to D as follows:

$$\overline{s} - t' < \frac{|\sigma(D)|}{|D|} < \overline{s} + t' \quad (10)$$

where $t' = \sqrt{\frac{(1-f_x)(\log 2 - \log \alpha)}{2|x|}}$. We can estimate the confidence interval for $|\sigma(D)|$ by using the following form that multiplied $|D|$ by Eq.10.

Lemma 2 $|D|(\overline{s} - t') < |\sigma(D)| < |D|(\overline{s} + t')$.

3.2 Confidence interval for mean of data slices

We derive the confidence interval for the mean of m of $\sigma(D)$ (expressed as $\mu(\sigma(D))$) by applying Lemma 2 to Eq.6. We can estimate the confidence interval for $\mu(\sigma(D))$ as follows by setting the size of population as $|\sigma(D)|$ and the size of samples as $|\sigma(x)|$:

$$\overline{\sigma(x)} - t'' < \mu(\sigma(D)) < \overline{\sigma(x)} + t'' \quad (11)$$

where $\overline{\sigma(x)}$ is the mean of $\sigma(x)$ and t'' is a value obtained by substituting $|\sigma(x)|$ and the range of f_x in Lemma 2 for $|x|$ and f_x in $t = \sqrt{\frac{(1-f_x)(max-min)^2(\log 2 - \log \alpha)}{2|x|}}$ as follows:

$$|x| \leftarrow |\sigma(x)|,$$

$$\frac{|\sigma(x)| - 1}{|D|(\overline{s} + t')} < f_x < \frac{|\sigma(x)| - 1}{|D|(\overline{s} - t')}$$

where the range of f_x is obtained by substituting $|\sigma(x)|$ and the range of $|\sigma(D)|$ in Lemma 2 for $|x|$ and $|D|$ in f_x respectively. Furthermore, t'' is removed from Eq.11 as follows. t'' has range $[t_{min}, t_{max}]$ because t is a function of f_x and f_x has range. When the size of population equals $|D|(\overline{s} - t')$ and $f_x = \frac{|\sigma(x)|-1}{|D|(\overline{s}-t')}$, t'' takes lowerbound t_{min}. When the size of population equals $|D|(\overline{s} + t')$ and $f_x = \frac{|\sigma(x)|-1}{|D|(\overline{s}+t')}$, t'' takes upperbound t_{max}. Therefore, Eq.11 is rewritten as follows.

Lemma 3 $\overline{\sigma(x)} - t_{max} < \mu(\sigma(D)) < \overline{\sigma(x)} + t_{max}$
where $t_{max} = \sqrt{\frac{(1-\frac{|\sigma(x)|-1}{|D|(\overline{s}+t')})(max-min)^2(\log 2 - \log \alpha)}{2|\sigma(x)|}}$.

3.3 Confidence interval for sum of data slices

We derive the confidence interval for the sum of m of $\sigma(D)$ (expressed as $\eta(\sigma(D))$) by multiplying $\mu(\sigma(D))$ by $|\sigma(D)|$. When $|\sigma(D)|$ takes the maximum of $|\sigma(D)|$ (i.e. $|D|(\overline{s}+t')$), t should equal t_{max} and $\mu(\sigma(D))$ takes the maximum of $\mu(\sigma(D))$ (i.e. $\overline{\sigma(x)} + t_{max}$) at the same time. Therefore, the maximum of $\eta(\sigma(D))$ is the value obtained by multiplying the maximum of $\mu(\sigma(D))$ by the maximum of $|\sigma(D)|$. On the other hand, when $|\sigma(D)|$ takes the minimum of $|\sigma(D)|$ (i.e. $|D|(\overline{s} - t')$), t should equal t_{min} but $\mu(\sigma(D))$ does not take the minimum of $\mu(\sigma(D))$ (i.e. $\overline{\sigma(x)} - t_{max}$). We define $\overline{\sigma(x)} - t_{opt}$ and $|D|(\overline{s} - t'_{opt})$ which are the values of $\mu(\sigma(D))$ and $|\sigma(D)|$ respectively when $\eta(\sigma(D))$ takes the minimum. We can estimate the confidence interval for $\eta(\sigma(D))$ by using the following form.

Lemma 4
$(\overline{\sigma(x)} - t_{opt})(|D|(\overline{s} - t'_{opt})) < \eta(\sigma(D)) < (\overline{\sigma(x)} + t_{max})(|D|(\overline{s} + t'))$
where $t_{opt} = \sqrt{\frac{(1-\frac{|\sigma(x)|-1}{|D|(\overline{s}-t'_{opt})})(max-min)^2(\log 2 - \log \alpha)}{2|\sigma(x)|}}$, *$t'_{opt}$ is the value within the range of $\overline{s} - t' < t_{opt} < \overline{s} + t'$ when the above $(\overline{\sigma(x)} - t_{opt})(|D|(\overline{s} - t'_{opt}))$ takes the minimum.*

4. EFFICIENT ALGORITHM FOR DATA SLICE SEARCH

The algorithm prunes the search space by employing the confidence interval and removes redundant query processing by applying multi-query optimization for evaluating queries over multiple data slices.

Pruning

One characteristic in identifying data slices that generate exceptional views is that most views generated from data slices would have a high similarity to the view generated from the whole data. Since those data slices cannot be within top-k of large deviation, it is preferable to prune the query evaluation for those data slices. To achieve this pruning, we only use samples and compute the confidence interval from samples for the data slice search. Specifically, we describe how we calculate deviations (Section 4.1), we estimate deviations from samples by computing the upper bound and lower bound of the confidence intervals for the aggregation of each data slice (Section 4.2). If the upper bound of the query result for data slice S is lower than the lower bound of the kth highest data slice, we can safely prune the query evaluation for the data slice S (Section 4.3).

Multi-query Optimization (Section 4.4)

In identifying data slices that generate exceptional views, we have to evaluate a given query over multiple data slices. This query processing is optimized by grouping the slicing queries for the data slices with respect to the same attributes and then by evaluating the grouped queries during a single scan.

4.1 Calculating deviations

We describe how to calculate the deviation between $q(D)$ and $q(S)$ (i.e. $deviation(q(D), q(S))$), which is computed by Euclidean Distance. We define ${}_gG_{a=f(m)}(S)$ in Eq.3 to the following form.

$$ {}_gG_{a=f(m)}(S) := \{f(\pi_m(\sigma_{g='Z'}(S)))|Z \in values(g)\} \quad (12)$$

where $values(g)$ is the set of the unique values of the group-by attribute g. In other words, ${}_gG_{a=f(m)}(S)$ selects subsets of S for each value Z of g, projects the measure attribute m of the data slices, and applies the aggregate function f. We can expand $deviation(q(D), q(S))$ in Definition 1 to the following form by using Eq.12.

$$deviation(\{f(\pi_m(\sigma_{g='Z'}(D)))|Z \in values(g)\},$$
$$\{f(\pi_m(\sigma_{g='Z'}(S)))|Z \in values(g)\}). \quad (13)$$

Furthermore, Eq.13 can be rewritten to the following form because *deviation* calculates Euclidean Distance.

$$\sqrt{\sum_{Z \in values(g)} (f(\pi_m(\sigma_{g='Z'}(D))) - f(\pi_m(\sigma_{g='Z'}(S))))^2}. \quad (14)$$

That is, $deviation(q(D), q(S))$ selects subset of D and S for each value $Z \in values(g)$, calculates the squares of the differences between the aggregation results of $\sigma_{g='Z'}(D)$ and $\sigma_{g='Z'}(S)$, and calculates the square root of the sum of the squares. When f is AVG or SUM, we normalize $q(D)$ and $q(S)$ so that the sum of $q(D)$ and the sum of $q(S)$ equal one.

4.2 Estimating deviations

We calculate $f(\pi_m(\sigma_{g='Z'}(D)))$ in Eq.14 beforehand and then estimate $f(\pi_m(\sigma_{g='Z'}(S)))$ by using the samples of S. We apply Lemma 2, 3 and 4 for the aggregate function COUNT, AVG and SUM, respectively. For example, we can express the confidence interval for the size of S (expressed as $range(|S|)$) by using Lemma 2 when f is COUNT.

$$range(|S|) = [|D|(\bar{s} - t'), |D|(\bar{s} + t')]. \quad (15)$$

We can calculate the upper bound of deviation between D and S by using the following form.

$$\sqrt{\sum_{Z \in values(g)} (max(f(\pi_m(\sigma_{g='Z'}(D))), range(\pi_m(\sigma_{g='Z'}(S)))))^2} \quad (16)$$

where $max(x, [a, b])$ is a function that takes a value and a range as arguments, and returns the value which takes the largest distance from x in range $[a, b]$. On the other hand, we can calculate the lower bound of the deviation, replacing max in Eq.16 to a function which returns the value which takes the smallest distance from x in the range $[a, b]$.

4.3 Pruning data slices

We prune the query evaluation for data slices that cannot be within top-k of large deviation. The pruning works as follows. We divide the dataset D into the samples and the rest. We evaluate given query q over all data slices $S \in \mathbb{S}$ for the samples. We compute the upper bound and lower bound of the deviations for the result of q. If the upper bound of the data slice S is lower than the lower bound of the kth highest data slice, we do not evaluate q over the data slice S for the rest of dataset. As a result, we can prune the query evaluation for the data slice that does not become the top-k result with a high probability.

4.4 Multi-query Optimization

By grouping the slicing queries for the data slices with respect to the same attributes, the grouped queries can be combined into a single query, which results in the reduction of wasteful computation and faster process. For example, when calculating the monthly sales of products bought by either males or females, combining two queries into a single query can be made. By grouping the slicing queries, the number of the slicing queries becomes equal to the number of dimension attributes B, and these grouped queries are executed with single scan.

5. PERFORMANCE EVALUATION

Our goal of experiments is to evaluate the efficiency and accuracy of our proposed algorithm in the data slice search. We compared the performance of naive approach (NO_OPT), with combining all slicing queries for data slices into single query regardless of different attributes (ONE_QUERY), with multi-query optimization (MULTI), and with the combination of multi-query optimization and data slice pruning optimizations by confidence interval (COMB).

5.1 Benchmark

We measured the performances by using various OLAP queries for real dataset.

Dataset: This dataset is sales data of supermarkets which is provided by Joint Association Study Group of Management Science. The dataset is composed of measure attributes (*price* and *number*) and dimension attributes (*store* (9), *time* (19), *sex* (3), *category_1* (8), *category_2* (25) and *category_3* (164)). The number in the parenthesis indicates the number of unique values of the attribute. The number of records is 103, 382, 016.

OLAP queries: We use *price* and *number* for aggregation and *store* and *time* for group-by.

Attributes for selecting data slices: We do not use the same attribute both for group-by operation and slicing data condition at the same time, because the result is not useful in such case; a single value is sliced out, group-by operation is applied, and then only a single value is aggregated.

All experiments were run on single machine with 16 GB RAM and 4 core Intel(R) Core(TM) i7-4702MQ processor. We evaluate our techniques on Microsoft SQL Server 2014. We used nonclustered columnstore indexes for efficiency. We removed extremely large values of measure attributes in advance for the purpose of data cleaning. We chose samples from the first part of the database. All experiments were repeated three times and the measurements were averaged.

5.2 Results

Figure 2 shows the latencies of top-10 data slices search for evaluating the OLAP queries over the dataset. The x-axis shows the OLAP queries and the label indicates aggregation attribute/group-by attribute. We observe that the multi-query optimization and data slice pruning are effective in improving the latencies for the dataset. In detail, the pruning data slice optimization effectively improves the latencies when the number of distinct values of group-by attribute is small, such as *store* (9). That is, Euclidean Distance between long sequences tends to be more similar than short sequences, so we have more opportunities of pruning the search space when the number of distinct values of

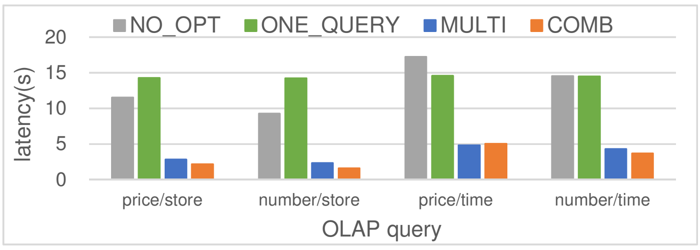

Figure 2: Latency experiments for the four approaches

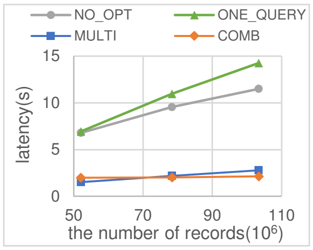

Figure 3: Scalability experiments

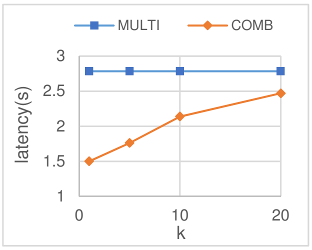

Figure 4: Effect of pruning at various k

group-by attribute is small. COMB takes longer time than MULTI when aggregate attribute is *price* and group-by attribute is *time*. This is because top-10 values of deviation was not so large and COMB prunes a small number of data slices. Although ONE_QUERY executes the smallest number of queries among the four approaches, it takes longer time than MULTI in all the OLAP queries. This result may be explained by the fact that, since we used nonclustered columnstoreindex, grouping slicing queries with respect to the same single attributes provide the best performance.

We made scalability experiments by varying the size of the datasets. As we can see in Figure 3, we observe that the latency for the case with the pruning optimization scales well and is constant even if the dataset size increases. This is because the latency with the pruning optimization mainly depends on k of top-k; even if the data size increases, required search space depends on top-k size.

We made experiments to verify the influence of k by varying the value of k. As we can see in Figure 4, we observe that the smaller the value of k, the better the efficiency of the pruning optimization. This is because the threshold used by pruning becomes large when the value of k becomes small, and pruning more data slices.

Note, for evaluating the accuracy of data slice pruning optimization, we validate that all the algorithms obtain the same top-k data slices in all the experiments.

6. RELATED WORK

Visual analytics tools such as Spotfire [1] and Polaris [11] have been introduced. These tools support the analysis of users by selecting the best visualization for a data set. The Users must chose data slices that they want to analyze.

Visual analytics tools such as Profiler [3], Vizdeck [4] and SEEDB [12, 13] have a function to select visualizations automatically. Profiler automatically detects exceptions in a data set. VizDeck has a function to depict visualizations of a data set on a dashboard. SEEDB recommend a visualization result of a data slice based on distance function for the given data slice. Sarawagi et al. [8] study an exploration of data cubes by using data mining techniques. They explore exceptional and interesting cells (say, Sales of clothes of March, 2014) in a cube. However, those techniques do not explore exceptional $q(D)$ (say, Monthly sales of clothes of 2014). Ordonez et al. [7] incorporate parametric statistical tests into analysis for interactive visualization of the OLAP cube. The visualization identifies significant differences between two similar cuboids.

7. CONCLUSION

We propose a framework that automatically identifies data slices that generate exceptional views for OLAP queries, and an efficient algorithm that optimizes the data slice search. The algorithm improves the performance by confidence interval and multi-query optimization. We used real dataset and validated that our algorithm improves the performance eight times faster than without the optimizations.

There are various types of future work for the data slice search. First, we extend the system to compute the deviation not only from the average of whole data, but also the deviation from the average of data clusters. Second, we introduce the hierarchy between attributes, such as categories, so that we can drill down and roll up the analysis results. Finally, we will extend the variations of data slices by permitting the combination of multiple attributes (conjunctive condition) for slicing condition.

8. ACKNOWLEDGMENTS

This work was supported by JSPS KAKENHI Grant-in-Aid for Scientific Research (C) 16k00154.

9. REFERENCES

[1] C. Ahlberg. Spotfire: An information exploration environment. *SIGMOD Rec*, 25(4), 1996.
[2] M. Buoncristiano, G. Mecca, E. Quintarelli, M. Roveri, D. Santoro, and L. Tanca. Database challenges for exploratory computing. *SIGMOD Rec*, 44(2), 2015.
[3] S. Kandel, R. Parikh, A. Paepcke, J. M. Hellerstein, and J. Heer. Profiler: Integrated statistical analysis and visualization for data quality assessment. In *Prc. AVI*, 2012.
[4] A. Key, B. Howe, D. Perry, and C. Aragon. Vizdeck: Self-organizing dashboards for visual analytics. In *Proc. SIGMOD*, 2012.
[5] A. Madraky. Data mining text book. 2012.
[6] K. Morton, M. Balazinska, D. Grossman, and J. Mackinlay. Support the data enthusiast: Challenges for next-generation data-analysis systems. *Proc. VLDB Endow*, 7(6), 2014.
[7] C. Ordonez, Z. Chen, and J. García-García. Interactive exploration and visualization of olap cubes. In *Proc. ACM DOLAP Workshop*, 2011.
[8] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *EDBT*, 1998.
[9] T. K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst*, 13(1), 1988.
[10] R. J. Serfling. Probability inequalities for the sum in sampling without replacement. *The Annals of Statistics*, 2(1), 1974.
[11] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional databases. *Commun. ACM*, 51(11), 2008.
[12] M. Vartak, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: Automatically generating query visualizations. *Proc. VLDB Endow*, 7(13), 2014.
[13] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: efficient data-driven visualization recommendations to support visual analytics. *Proc. VLDB Endow*, 8(13), 2015.