Steerable Self-Driving Data Visualization

Yuyu Luo, Xuedi Qin, Chengliang Chai, Nan Tang, Guoliang Li, and Wenbo Li

Abstract—In this work, we present a self-driving data visualization system, called DEEPEYE, that automatically generates and recommends visualizations based on the idea of *visualization by examples.* We propose effective visualization recognition techniques to decide which visualizations are meaningful and visualization ranking techniques to rank the good visualizations. Furthermore, a main challenge of automatic visualization system is that the users may be misled by blindly suggesting visualizations without knowing the user's intent. To this end, we extend DEEPEYE to be easily steerable by allowing the user to use *keyword search* and providing click-based *faceted navigation*. Empirical results, using real-life data and use cases, verify the power of our proposed system.

Index Terms—Data visualization, visualization recommendation, data exploration, keyword search, faceted navigation

1 INTRODUCTION

NOWADAYS, the ability to create good visualizations has shifted from a nice-to-have skill to a must-have skill for all data analysts [1]. The overwhelming choices of data visualization tools (e.g., Tableau and Qlik) have allowed users to create good visualizations, *only if* the users know their data well. Ideally, the users need tools to automatically recommend visualizations, so they can pick interesting ones.

Technically speaking, "interesting" charts can be defined from three angles: (1) *Deviation-based*: a chart that is dramatically different from the other charts (e.g., SeeDB [2]); (2) *Similarity-based*: charts that show similar trends *w.r.t.* a given chart (e.g., zenvisage [3]); and (3) *Perception-based*: visualizations that can tell compelling stories, from understanding the data, without being compared with other references.

> "If I had an hour to solve a problem I'd spend 55 minutes thinking about the problem and 5 minutes thinking about solutions."
>
> — Albert Einstein —

Although (1) "statistical deviation" and (2) "similarity" can be quantified formally, our 55 minutes thought is to study (3) because one fundamental request from users is to find not only eye-catching but also informative charts.

Example 1. Consider a real-world table about *flight delay statistics of Chicago O'Hare International (Jan – Dec, 2015)*, with an excerpt in Table 1 (https://www.bts.gov). Naturally, the Bureau of Transportation Statistics wants to visualize some valuable insights/stories of the data. Fig. 1 shows sample visualizations DEEPEYE considers for the entire table.

i) Fig. 1a is a scatter plot, with x-axis: *Departure Delay (min)*, y-axis: *Arrival Delay (min)*, and plots grouped (and colored) by *Carrier*. It shows clearly the arrival delays *w.r.t.* departure delays for different carriers, e.g., the carrier OO is bad due to its long departure and arrival delays.
ii) Fig. 1b is a stacked bar chart, with x-axis: *Scheduled* binned by month, y-axis: the number of *Passengers* in each month that is stacked by *Destination City Name*. It shows the number of passengers travelled to *where* and *when*.
iii) Fig. 1c is a line chart, with x-axis: *Scheduled* binned by hour (i.e., the rows with the same hour are in the same bucket), y-axis: the average of *Departure Delay (min)*. It shows *when* is likely to have more departure delays, e.g., it has long delays in late afternoon.
iv) Fig. 1d is a line chart, with x-axis: *Scheduled* binned by date, y-axis: the average of *Departure Delay (min)*. It shows the range of delays, no trend.

Self-driving Data Visualization. From the user perspective, users want data visualization systems to automatically discover compelling stories of the data, which is also known as visualization recommendation systems. Not surprisingly, there have been proposals for such systems [4], which focus on automatically discovering "interesting" visualizations from different criteria, such as relevance, surprise, non-obviousness, diversity and coverage. However, as pointed out by [5], these systems may mislead the user, by generating visualizations that might be *worse than nothing*.

Steerable Self-Driving Data Visualization. In order to better navigate the discovery process for finding compelling stories, users need to steer in a simple way, e.g., search visualizations by keyword or click-based faceted navigation. Building such a system faces several challenges.

I) *Capturing Human Perception*. How to quantify that which visualization is good, better, or the best?

- *Y. Luo, X. Qin, C. Chai, G. Li, and W. Li are with the Department of Computer Science, Tsinghua University, Beijing 100084, China. E-mail: {luoyy18, qxd17, chaicl15, li-wb17}@mails.tsinghua.edu.cn, liguoliang@tsinghua.edu.cn.*
- *N. Tang is with Qatar Computing Research Institute, Hamad Bin Khalifa University, Doha, Qatar. E-mail: ntang@hbku.edu.qa.*

Manuscript received 2 Oct. 2019; revised 23 Jan. 2020; accepted 28 Feb. 2020. Date of publication 0 . 0000; date of current version 0 . 0000. (Corresponding authors: Guoliang Li and Chengliang Chai.) Recommended for acceptance by M. Wang. Digital Object Identifier no. 10.1109/TKDE.2020.2981464

TABLE 1
An Excerpt of Flight Delay Statistics

Scheduled	Carrier	Destination City Name	Departure Delay (min)	Arrival Delay (min)	Passengers
01-Jan 00:05	UA	New York	−4	1	193
01-Jan 04:00	AA	Los Angeles	0	−2	204
01-Jan 06:13	MQ	San Francisco	7	−11	96
01-Jan 07:33	OO	Atlanta	11	−2	112
…	…	…	…	…	…

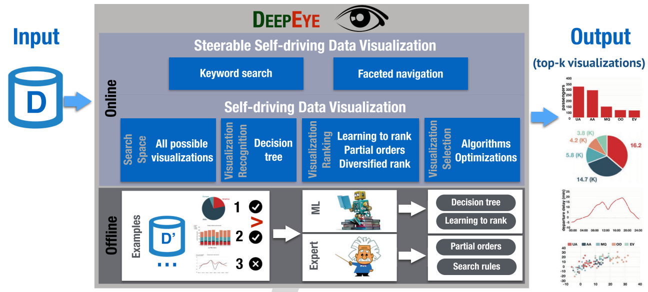


Fig. 2. Framework overview of DeepEye.

II) *Large Search Space.* Sometimes, visualizing a dataset *as-is* cannot produce any interesting output. Appearances can, however, be deceiving, when the stories reside in the data after being transformed, such as selections for columns, groups, and aggregations – these create a huge search space.

III) *Lack of Ground Truth.* A benchmark or the ground truth of a given dataset is often unavailable.

Intuitively, there are two ways of handling Challenge (I): (A) Learning from examples–there are plenty of generic priors to showcase great visualizations. (B) Expert knowledge, e.g., a bar chart with more than 50 bars is clearly bad. Challenge (II) is a typical database optimization problem that techniques such as pruning and other optimizations can play a role. For Challenge (III), fortunately, there are online tables accompanied with well-designed charts, which are treated as good charts. Besides, we also ask researchers to manually annotate to create "ground truth".

Contributions. We have built DeepEye and made it available as a web service (http://deepeye.tech), with the following notable contributions.

Self-driving Data Visualization. (1) Visualization Recognition: We propose to capture human perception about the goodness of visualizations by learning from existing examples, which differs from other visualization recommendation systems that are either deviation-based (e.g., SeeDB [2]), or similarity-based (e.g., zenvisage [3]). *(2) Visualization Ranking:* We propose effective ranking techniques to rank the visualizations, including learning-to-rank, partial order, and diversity. *(3) Visualization Selection:* We present a graph based approach, as well as rule-based optimizations to efficiently compute top-k visualizations by filtering bad visualizations that do not need to be considered.

Steerable Self-driving Data Visualization. (4) Keyword Search: It allows the users to create good visualizations via keyword search. *(5) Faceted Navigation:* It also provides a click-based faceted navigation (for example, similar trend, different trend, varying x/y-axis) such that the user can easily navigate the potentially huge search space.

Experiments. (6) We conduct experiments using real-world datasets, and visualization use cases, to show that DeepEye can efficiently discover interesting visualizations.

Differences With the Conference Version. *This work extends our conference version [6] with the following new contributions: (1) We propose to select diversified top-k visualizations since there may be many similar visualizations showing redundant information. We prove that the problem is NP-hard and present an efficient and effective algorithm to solve it (Section 6); (2) We design and implement the keyword search and faceted navigation module, which enables the user to interact with our system easily (Section 7); (3) We develop a novel "steerable self-driving" data visualization system that is available online (http://deepeye.tech); and (4) We conduct a new empirical evaluation to verify the effectiveness and efficiency of DeepEye (Section 8).*

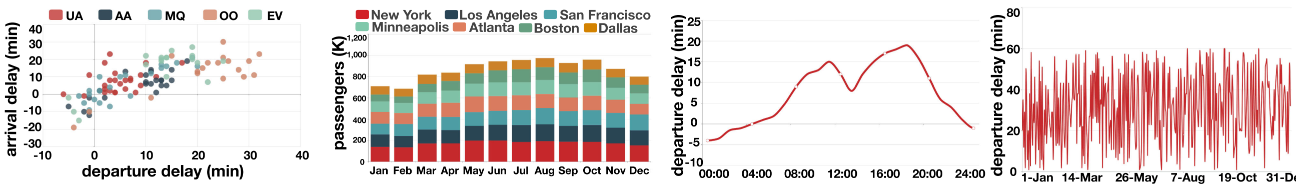
2 System Overview and Background

2.1 An Overview of DeepEye

An overview of DeepEye is given in Fig. 2, which consists of an offline component and an online component.

Offline Component relies on examples–good visualizations, bad visualizations, and ranks between visualizations– to train two ML models: a binary classifier (e.g., a *decision tree*) to determine whether a given dataset and an associated visualization is good or not, and a *learning-to-rank* model that ranks visualizations (see Section 3 for more details). Alternatively, experts may specify partial order as rules based on their knowledge to rank visualizations, which will be discussed in Section 4.

Online Component identifies all possible visualizations, uses the trained classifier to determine whether a visualization is good or not, employs either the learning-to-rank model or expert provided partial orders to select (diversified) top-k visualizations (see more details in Sections 4, 5, and 6). In order to better navigate the discovery process, naturally, users can pose some keywords to get some visualizations or navigate the visualizations via faceted navigation (more details in Section 7).



(a) Flight delay distribution (b) Monthly #-passengers, by dest. (c) Flight delay *w.r.t.* scheduled (d) Flight delay *w.r.t.* dates

Fig. 1. Sample visualizations for the flight delay statistics.

VISUALIZE	TYPE ($\in$ {bar, pie, line, scatter})
SELECT	X', Y' ($X' \in \{X, \texttt{BIN}(X)\}$, $Y' \in \{Y, \texttt{AGG}(Y)\}$)
FROM	D
TRANSFORM	X (using an operator $\in$ {BIN, GROUP})
ORDER BY	X', Y'

Fig. 3. Visualization language (two columns).

2.2 Preliminaries

We consider a relational table D, defined over the schema $\mathcal{A}(A_1, \ldots, A_m)$ with m attributes (or columns).

We study four widely used visualization types: bar charts, line charts, pie charts, and scatter charts.

We consider the following three types of data operations.

1. Transform. It aims to transform the values in a column to new values based on the following operations.

- *Binning* partitions the numerical or temporal values into different buckets:
  - *Temporal values* are binned by minute, hour, day, week, month, quarter, year, whose data type can be automatically detected based on the attribute values.
  - *Numerical values* are binned based on consecutive intervals, e.g., bin1$[0, 10)$, bin2$[10, 20)$, $\ldots$; or the number of targeted bins, e.g., 10 bins.
- *Grouping* groups values based on categorical values.

2. Aggregation. Binning and grouping are to categorize data together, which can be consequently interpreted by aggregate operations, SUM (sum), AVG (average), and CNT (count), for the data that falls in the same bin or group. Hence, we consider three aggregation operations: $\texttt{AGG} = \{\texttt{SUM}, \texttt{AVG}, \texttt{CNT}\}$.

3. Order By. It sorts the values based a specific order. Naturally, we want some scale domain, e.g., x-scale, to be sorted for easy understanding of some trend. Similarly, we can also sort y-scale to get an order on the y-axis.

2.3 Visualization Language

To facilitate our discussion, we define a simple language to capture all possible visualizations studied in this paper. For simplicity, we first focus on visualizing two columns, as shown in Fig. 3. Each query contains three mandatory clauses (*VISUALIZE*, *SELECT*, and *FROM* in bold) and two optional clauses (*TRANSFORM* and *ORDER BY* in italic). They are further explained below.

▷ *VISUALIZE*: specifies the visualization type

▷ *SELECT*: extracts the selected columns

- X'/Y' relates to X/Y: X' just X, grouping values in X, or binning values in X (e.g., by hour); Y' is either Y or the aggregation values (e.g., $\texttt{AGG} = \{\texttt{SUM}, \texttt{AVG}, \texttt{CNT}\}$) after transforming X

▷ *FROM*: the source table

▷ *TRANSFORM*: transforms the selected columns

- Binning
  - BIN X BY {MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR}.
  - BIN X INTO N, where N is the targeted #-bins.
  - BIN X BY UDF(X), where UDF is a user-defined function, e.g., splitting X by given values (e.g., 0).
- Grouping: GROUP BY X

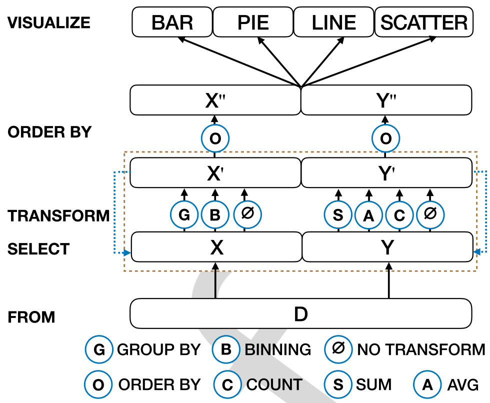


Fig. 4. Search space for two columns.

▷ *ORDER BY*: sorts the selected column, X' or Y'

Each visualization query Q over D, denoted by $Q(D)$, will produce a chart, which is also called a *visualization*.

Example 2. One sample visualization query $Q_1(D)$ is given below, which is used to visualize Fig. 1c.

VISUALIZE	line	
SELECT	*Scheduled*, AVG(*Departure Delay (min)*)	
FROM	*TABLE I*	$Q_1(D)$
BIN	*Scheduled BY HOUR*	
ORDER BY	*Scheduled*	

Search Space. Given a dataset D, there exist multiple visualizations. All possible visualizations form our search space, which is shown in Fig. 4 for two columns.

▷ *SELECT* can take any ordered column pairs (i.e., XY and YX are different), which gives $m \times (m-1)$.

▷ *TRANSFORM* can either group by X, bin X (we have 9 cases, e.g., by minute, hour, day, week, month, quarter, year, default buckets and UDF), or do nothing; and aggregate Y using different operations. Thus there are $(1 + 9 + 1) \times 4 = 44$ cases for each column pair.

▷ *ORDER BY* can order either column X', column Y', or neither: these give 3 possibilities. Note that we cannot sort both columns at the same time.

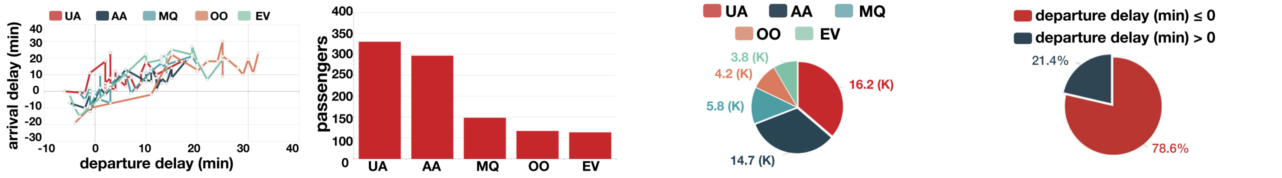
Together with the four visualization types, the number of all possible visualizations for two columns is: $m \times (m-1) \times 44 \times 4 \times 3 = 528\ m(m-1)$, which is fairly large for wide tables (i.e., the number of columns m is large).

Remark. As surveyed by [7], real users strongly prefer bar, line, and pie charts. In particular, the percentages of bar, line and pie charts are 34, 23, and 13 percent respectively; and the total percentage of the three types is around 70 percent. Thus this work focuses on these chart types and leaves supporting other chart types as a future work.

3 Machine Learning-Based Visualization Recognition, Ranking, and Selection

A natural way to capture human perception is by learning from examples, through machine learning.

Features. It is known that the performance of machine learning methods is heavily dependent on the choice of



(a) Flight delay distribution (b) Avg. #-passengers by carriers (c) Total #-passengers *w.r.t.* carriers (d) Proportions of departure delay

Fig. 5. More sample visualizations for the flight delay statistics.

features (or representations) on which they are applied. Much of our effort goes into this feature engineering to support effective machine learning. We identify the following features $\mathbf{F}$.

1) The number of distinct values in column X, $d(X)$.
2) The number of tuples in column X, $|X|$.
3) The ratio of unique values in column X, $r(X) = \frac{d(X)}{|X|}$.
4) The $\max(X)$ and $\min(X)$ values in column X.
5) The data type $\mathbf{T}(X)$ of column X:
   - *Categorical:* contains only certain values, e.g., carriers.
   - *Numerical:* contains only numerical values, e.g., delays.
   - *Temporal:* contains only temporal data, e.g., date.
   - ◦ We also use abbreviations: `Cat` for categorical, `Num` for numerical, and `Tem` for temporal.
6) The correlation of two columns, $c(X, Y)$, is a value between -1 and 1. The larger the value is, the higher correlation the two columns have. We consider linear, polynomial, power, and log correlations. We take the maximum value in these four cases as the correlation between X and Y.
7) The visualization type: bar, pie, line, or scatter charts.

For two columns X, Y, we have the above features (1–5) for each column, which gives $6 \times 2 = 12$ features; together with (6) and (7), we have a feature vector of 14 features.

Visualization Recognition. The first task is, given a column combination of a dataset and a specified visualization type, to decide whether the output (i.e., the visualization node) is good or bad. Hence, we just need a binary classier, for which we use decision tree. We have also tested Bayes classifier and SVM, and the decision tree outperforms SVM and Bayes (see Section 8 for empirical comparisons).

Visualization Ranking. The other task is, given two visualization nodes, to decide which one is better, for which we use a *learning-to-rank* [8] model, which is an ML technique for training the model in a ranking task, which has been widely employed in Information Retrieval (IR), Natural Language Processing (NLP), and Data Mining (DM).

Roughly speaking, it is a supervised learning task that takes the input space $\mathcal{X}$ as lists of feature vectors, and $\mathcal{Y}$ the output space consisting of grades (or ranks). The goal is to learn a function $F(\cdot)$ from the training examples, such that given two input vectors $\mathbf{x_1}$ and $\mathbf{x_2}$, it can determine which one is better, $F(\mathbf{x_1})$ or $F(\mathbf{x_2})$. We used the LambdaMART algorithm [9].

Visualization Selection. Learning-to-rank model can be used directly for the visualization selection problem: given a set of visualization nodes (and their features vectors) as input, outputs a ranked list.

Remarks. Using ML models as *black-boxes* has two shortcomings. (1) They may not capture human perception as precise as experts in some aspects, e.g., there are not enough examples for comparing visualizations for different columns. (2) It is hard to improve search performance of black-boxes. Naturally, expert knowledge should be leveraged when it can be explicitly specified.

4 Partial-Order-Based Selection

Computing top-k visualizations requires a *ranking* for all possible visualizations. Ideally, we expect a *total order* of visualizations such that the top-k can be trivially identified. However, it is hard to define a total order, because two visualizations may not be directly comparable. A more feasible way, from the user perspective, is to specify *partial orders* for comparable visualizations. Afterwards, we can obtain a directed graph representing the partially ordered set of visualizations (*a.k.a.* a Hasse diagram).

We first discuss the ranking principle (Section 4.1), and define partial orders (Section 4.2). We then present an algorithm to compute top-k visualizations based on the partial order (Section 4.3). We also propose a hybrid method by combing learning-to-rank and partial order (Section 4.4).

4.1 Visualization Ranking Principle

Definition 1 [Visualization Node]. *A visualization node v consists of the original data X, Y, the transformed data X', Y', features $\mathbf{F}$, and the visualization type $\mathbf{T}$.*

Given two visualization nodes v and u, we use X_1/Y_1 (resp. X_2/Y_2) to denote the two columns of v (resp. u), and X_1'/Y_1' (resp. X_2'/Y_2') to denote the transformed columns.

Example 3. Fig. 5 shows more visualizations of Table 1. We take 2 visualizations in Figs. 1 and 3 visualizations in Fig. 5 to illustrate the definition of *visualization node*, which are shown in Table 2.

We consider three cases, based on different possibilities of columns shared between two visualizations.

Case 1. $X_1 = X_2$ and $Y_1 = Y_2$: they have the same original data. Again, we consider two cases, (I) the same transformed data (i.e., $X_1' = X_2'$ and $Y_1' = Y_2'$) and (II) different transformed data ($X_1' \neq X_2'$ or $Y_1' \neq Y_2'$).

I) $X_1' = X_2'$ and $Y_1' = Y_2'$: we adopt the techniques from the visualization community to rank visualizations [10], [11].

i) X_1' and X_2' are categorical: pie/bar charts are better than scatter/line charts, because the latter two focus on the trend and correlation between X and Y.

TABLE 2
Example of Visualization Node

Vis. Node	Attributes		
	Data	F	T
Fig. 1(c)	X = *Scheduled* Y = *Departure Delay (min)* X' = BIN (*Scheduled*) BY HOUR Y' = AVG (*Departure Delay (min)*)	$\|X\| = \|Y\| = 99527$ $\|X'\| = \|Y'\| = 24$ $d(X') = 24$ $d(Y') = 18$ $c(X', Y') = 0.43$	Line
Fig. 1(d)	X = *Scheduled* Y = *Departure Dela(min)y* X' = BIN (*scheduled*) BY DAY Y' = AVG (*Departure Delay (min)*)	$\|X\| = \|Y\| = 99527$ $\|X'\| = \|Y'\| = 365$ $d(X') = d(Y') = 365$ $c(X', Y') = 0.14$	Line
Fig. 5(b)	X = *Carrier* Y = *Passengers* X' = GROUP BY (*Carrier*) Y' = AVG (*Passengers*)	$\|X\| = \|Y\| = 99527$ $\|X'\| = \|Y'\| = 5$ $d(X') = d(Y') = 5$ $c(X', Y') = null$	Bar
Fig. 5(c)	X = *Carrier* Y = *Passengers* X' = GROUP BY (*Carrier*) Y' = SUM (*Passengers*)	$\|X\| = \|Y\| = 99527$ $\|X'\| = \|Y'\| = 5$ $d(X') = d(Y') = 5$ $c(X', Y') = null$	Pie
Fig. 5(d)	X = *Departure Delay (min)* Y = *Departure Delay (min)* X' = BIN (*Departure Delay (min)*) BY ZERO Y' = CNT (*Departure Delay (min)*)	$\|X\| = \|Y\| = 99527$ $\|X'\| = \|Y'\| = 2$ $d(X') = d(Y') = 2$ $c(X', Y') = null$	Pie

  - If Y_1' and Y_2' are obtained by AVG, then bar charts are better, because pie charts are best used when making part-to-whole comparisons but we cannot get part-to-whole ratio by the AVG operation.
  - It would be better to use bar charts if there are many categories (for example, ≥ 10), because it is hard to put many categories in a single pie chart.
  - If $\min(Y_1') < 0$, pie charts are not applicable.

ii) X_1' and X_2' are numerical: scatter/line charts are better than pie/bar charts.

  - If there is a correlation between X' and Y', then scatter charts are better, because the scatter plot is simply a set of data points plotted on an x and y axis to represent two sets of variables. The shape those data points create tells the story, most often revealing correlation (positive or negative) in a large dataset.
  - If there is no correlation, line charts are better, because it shows time-series relationships with continuous data. It allows a quick assessment of acceleration (lines curving upward), deceleration (lines curving downward), and volatility (up/down frequency).

Observed from Case 1(I), we need to consider a factor to rank different charts. *Factor 1 - The matching quality between the data and charts*: whether the charts can visualize the inherent features of the data, e.g., trend, correlation.

II) $X_1' \neq X_2'$ or $Y_1' \neq Y_2'$: they have different transformed data. Typically, the smaller the cardinality of the transformed data, the better.

We consider another factor from Case 1(II). *Factor 2 - The quality of transformation operations*: whether the transformation operators make sense.

Case 2: $X_1 \neq X_2$ or $Y_1 \neq Y_2$, and $\{X_1, Y_1\} \cap \{X_2, Y_2\} \neq \emptyset$: They share a common column. Intuitively, for different columns, a user is more interested in visualizing an "important column". We consider another factor based on Case 2. *Factor 3 - The importance of a column:* whether it is important to visualize.

Case 3: $\{X_1, Y_1\} \cap \{X_2, Y_2\} = \emptyset$: they do not share common attributes. It is hard to directly compare two visualizations. Our hope is to use the transitivity of partial orders, based on the above three factors, to rank them.

4.2 Partial Order

Now we are ready to formally introduce our methodology to quantify visualizations so as to (partially) rank them, based on the above factors.

Factor 1 - The matching quality between data and chart $\mathbf{M}(v)$. It is to quantify the "goodness" of this visualization for the data and visualization type in v, with four cases.

i) Pie Chart. If the aggregation function is AVG, i.e., $Y' = \mathtt{AVG(Y)}$, then the pie chart doesn't make sense as pie charts are best used when making part-to-whole comparisons, and we set the value as 0. If there is only one distinct value $|d(X)| = 1$, we cannot get much information from the pie chart and thus we set the value as 0. If there are a small number of values, the pie chart has large significance, and we set the value as 1. If there are many distinct values (e.g., > 10), the significance of the pie chart will decrease [12], and we set the value as $\frac{10}{|d(X)|}$. In addition, if Y values are similar, the pie chart has no much meaning, and we prefer the Y values have large difference. It is defined below.

$$\mathbf{M}(v) = \begin{cases} 0 & \begin{array}{l} |d(X)| = 1 \\ \text{or } \min(Y') < 0 \\ \text{or } Y' = \mathtt{AVG(Y)} \end{array} \\ \sum_{y \in Y} -p(y)\log(p(y)) & 2 \leq |d(X)| \leq 10 \\ \frac{10}{|d(X)|}\sum_{y \in Y} -p(y)\log(p(y)) & |d(X)| > 10 \end{cases}. \tag{1}$$

ii) Bar Chart. The significance of bar chart is similar to the pie chart and the difference is that bar charts can tolerate large $|d(X)|$ (e.g., > 20) [11] and has no requirement that Y values have diverse values, and compute the score as below.

$$\mathbf{M}(v) = \begin{cases} 0 & |d(X)| = 1 \\ 1 & 2 \leq |d(X)| \leq 20 \\ \frac{20}{|d(X)|} & |d(X)| > 20 \end{cases}. \tag{2}$$

iii) Scatter Chart. We visualize scatter chart only if X, Y are highly correlated. Thus we can set the value as $c(X, Y)$.

$$\mathbf{M}(v) = c(X, Y). \tag{3}$$

iv) Line Chart. We visualize line charts if X is temporal or numerical columns. We want to see the trend of the Y values. Thus we use the trend distribution to

$$\mathbf{M}(v) = \mathsf{Trend}(Y), \tag{4}$$

where $\mathsf{Trend}(Y) = 1$ if Y follows a distribution, e.g., linear distribution, power-law distribution, log distribution or exponential distribution; otherwise, $\mathsf{Trend}(Y) = 0$.

Normalized Significance. Since it is hard to compare the significance of different charts, we normalize the significance for each chart and compute the score as below.

$$\mathbf{M}(v) = \frac{\mathbf{M}(v)}{\mathsf{maxM}}, \tag{5}$$

where maxM is the maximal score among all the nodes with the same chart with v.

Factor 2 - The quality of transformations $\mathbf{Q}(v)$. If the transformed data has similar cardinality with the original data, then the transformation is bad. Thus we use the ratio of the

TABLE 3
Factors of Visualization Node

	$\mathbf{M}(v)$	$\mathbf{Q}(v)$	$\mathbf{W}(v)$
Fig. 1c	1.00	0.99976	0.89
Fig. 1d	0	0.99633	0.52
Fig. 5b	0.73	0.99995	0.36
Fig. 5c	1.00	0.99995	0.36
Fig. 5d	0.36	0.99998	0.55

cardinality of the transformed data to the cardinality of the original data to evaluate the quality, i.e., $\frac{|X'|}{|X|}$, and the smaller the better. Thus we compute the value as

$$\mathbf{Q}(v) = 1 - \frac{|X'|}{|X|}. \tag{6}$$

Factor 3 - The importance of columns $\mathbf{W}(v)$. We first define the importance of a column $X, \mathbf{W}(X)$, which is the ratio of the number of candidate visualizations containing column X to the number of candidate visualizations. Note that the candidate visualizations are those visualizations considered for partial order. Clearly, the more important a column is, the better to visualize the chart with the column. Thus we compute the node weight by summing the weight of all columns in the node.

$$\mathbf{W}(v) = \sum_{X \in v} \mathbf{W}(X). \tag{7}$$

We normalize $\mathbf{W}(v)$ into $[0, 1]$ as below.

$$\mathbf{W}(v) = \frac{\mathbf{W}(v)}{\mathsf{max}\mathbf{W}}, \tag{8}$$

where $\mathsf{max}\mathbf{W}$ is the maximal $\mathbf{W}(v)$ among all nodes.

Example 4. Given Table 1, we get 44 candidate visualizations after visualization recognition. There are 27 candidate visualizations with column *Scheduled* and 12 candidate visualizations with column *Departure Delay (min)*. Thus the $\mathbf{W}(v)$ of visualization node *Fig. 1c* is $\frac{27}{44} + \frac{12}{44} = 0.89$.

Given two visualization nodes u, v, if u is better than v on every factor, i.e., $\mathbf{M}(u) \geq \mathbf{M}(v)$, $\mathbf{Q}(u) \geq \mathbf{Q}(v)$, $\mathbf{W}(u) \geq \mathbf{W}(v)$, then intuitively, u should be better than v. Based on this observation, we define a partial order.

Definition 2 [Partial Order]. *A visualization node u is better than a node v, denoted by $u \succeq v$, if $\mathbf{M}(u) \geq \mathbf{M}(v)$, $\mathbf{Q}(u) \geq \mathbf{Q}(v)$, $\mathbf{W}(u) \geq \mathbf{W}(v)$. Moreover, u is strictly better than v, denoted by $u \succ v$, if any of the above "$\geq$" is "$>$".*

Example 5. Based on the visualizations node in Table 2, we calculate the $\mathbf{M}(v)$, $\mathbf{Q}(v)$ and $\mathbf{W}(v)$ and get Table 3. Table 3 shows the score of three factors that influence partial order of the visualization nodes. Table 4 shows the partial order of the five visualization nodes in the Table 3.

TABLE 4
Example of Partial Order

Fig.	*1c*	*1d*	*5b*	*5c*	*5d*
1c	$\succeq$	$\succ$	$\backslash$	$\backslash$	$\backslash$
1d	$\backslash$	$\succeq$	$\backslash$	$\backslash$	$\backslash$
5b	$\backslash$	$\backslash$	$\succeq$	$\backslash$	$\backslash$
5c	$\backslash$	$\backslash$	$\succ$	$\succeq$	$\backslash$
5d	$\backslash$	$\succ$	$\backslash$	$\backslash$	$\succeq$

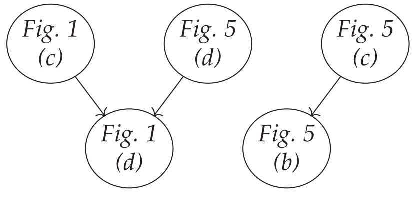


Fig. 6. Example of partial order graph.

Note that, comparing different types of charts is a hard problem. However, it is common in many search engines, e.g., Google returns ranked results with a mixture of videos, images and webpages. Consequently, any metric is heuristic. As will be verified empirically in Section 8, our normalized scores for different types of charts perform well in practice.

4.3 Partial Order-Based Visualization Selection

Given a table, we first enumerate all visualizations, and use the trained binary classifier to generate the candidate visualizations. Then for every pair of candidate visualizations, we check whether they conform to the partial order. If yes, we add a directed edge. Thus we get a graph $\mathbf{G}(V, E)$, where V is all visualization nodes and E indicates visualization pairs that satisfy partial orders. The weight between u and v, where $u \succeq v$, is defined as $w(u, v)$

$$w(u, v) = \frac{\mathbf{M}(u) - \mathbf{M}(v) + \mathbf{Q}(u) - \mathbf{Q}(v) + \mathbf{W}(u) - \mathbf{W}(v)}{3}. \tag{9}$$

We illustrate by examples about how to rank visualization nodes based on the graph.

Example 6. In Table 4, *Fig. 1c* $\succ$ *Fig. 1d*, so there is a directed edge between visualization node *Fig. 1c* and visualization node *Fig. 1d*. And the weight is $((1.00 - 0) + (0.99976 - 0.99633) + (0.89 - 0.52))/3 = 0.4578$. Based on the partial order in Table 4, we can construct the graph $\mathbf{G}$ using the visualization nodes *Figs. 1c*, *1d*, *5b*, *5c*, and *5d*, which is shown in Fig. 6.

Efficiently Construct the Graph $\mathbf{G}$. It is expensive to enumerate every node pair to add the edges. To address this issue, we propose a quick-sort-based algorithm. Given a node v, we partition other nodes into three parts: those better than v ($v^{\prec}$), those worse than v ($v^{\succ}$), and others ($v^{\not\prec\not\succ}$). Then for each node in $u \in v^{\prec}$ (or $v^{\succ}$), we do not need to compare with nodes in $v^{\succ}$ (or $v^{\prec}$). Thus we can prune many unnecessary pairs. We can also utilize the range-tree-based indexing method to efficiently construct the graph [13].

Rank Visualization Nodes based on $\mathbf{G}$. A straightforward method uses topology sorting to get an order of the nodes. It first selects the node with the least number of in-edges, and take it as the best node. Then it removes the node and selects the next node with the least number of in-edges. Iteratively, we can get an order.

However this method does not consider the weights on the edges. To address this issue, we propose a weight-aware approach. We first assign each node with a *visualization score* $S(v)$. The larger the $S(v)$ is, the better the visualization v is.

$\triangleright$ If a visualization node v without out-edge, $S(v) = 0$, else $S(v) = \sum_{(v,u)\in E}(w(v,u) + S(u))$, where $w(v,u)$ is the weight of edge (v,u).

Afterwards, we can select the k nodes with the largest scores. Algorithm 1 shows the pseudo code.

Algorithm 1. Partial Order-Based Selection

```
Input: V = {v_1, v_2, ..., v_n};
Output: top-k visualization nodes;
for each node v in V do
    Compute M(v), Q(v), W(v);
    Partition V − {v} into three parts: V^≺, V^≻, V^⊀⊁;
    Prune unnecessary pairs according to partitions;
Construct G(V, E) based on range-tree-based indexing;
ComputeNodeScore(v = root of V);
return top-k nodes v with largest weights S(v);
```

Function. ComputeNodeScore

```
Input: v
Output: S(v)
if outdegree(v) = 0 then
    return S(v) = 0
else
    for each(v, u) in E do
        ComputeNodeScore(u);
    return S(v) = Σ_{(v,u)∈E} (w(v, u) + S(u));
```

Example 7. We use Fig. 6 to illustrate this process. Suppose we aim to get the top-3 visualization nodes in this case. Fig. 6 shows the graph constructed by the visualization nodes in Table 4. Since the out-edges of *Figs. 5b* and *1d* are 0, the scores of *Figs. 5b* and *1d* are 0. Next, we show how to compute the scores of the other three nodes.

The weights of edges are: w(*Fig. 1c, Fig. 1d*) = 0.4578, w(*Fig. 5d, Fig. 1d*) = 0.1312, w(*Fig. 5c, Fig. 5b*) = 0.09.

The scores of the visualization nodes are:
S(*Fig. 1c*) $=$ w(*Fig. 1c, Fig. 1d*) + S(*Fig. 1d*) = 0.4578,
S(*Fig. 5d*) $=$ w(*Fig. 5d,Fig. 1d*) + S(*Fig. 1d*) = 0.1312,
S(*Fig. 5c*) $=$ w(*Fig. 5c, Fig. 5b*) + S(*Fig. 5b*) = 0.09.

Therefore, the top-3 visualization nodes are *Figs. 1c*, *5d*, and *5c*.

4.4 Hybrid Ranking Method

Learning-to-rank works well when there are sufficient good examples (i.e., supervised). Partial order works well when the experts have enough expertise to specify domain knowledge (i.e., unsupervised). We propose a hybrid method HybridRank to linearly combine these two methods as follows. Consider a visualization v. Suppose its ranking position is l_v by learning-to-rank and its ranking position is p_v by partial order. Then we assigns v with a score of $l_v + \alpha p_v$, where α is the preference weight which can be learned by some labelled data, and rank the visualizations by the score.

5 Optimizing Partial Order-Based Method

A closer look at the process of visualization enumeration (i.e., the search space) suggests that some visualizations should not be considered at all–those visualizations that human will never generate or consider, even if they have unlimited budget (or time). In order to directly prune these bad visualizations, we define rules to capture "meaningful" operations (Section 5.1). We then present algorithms that utilize these rules to compute top-k visualizations (Section 5.2) and discuss how to generate rules (Section 5.3).

5.1 Decision Rules for Meaningful Visualizations

We are ready to present the rules that can (possibly) generate meaningful visualizations from three perspectives: (1) transformation rules: whether a grouping or binning operation is useful; (2) sorting rules: whether a column should be sorted; and (3) visualization rules: whether a certain type of visualization is right choice. These rules use the features (or data representations) discussed in Section 3.

1. Transformation Rules. We first consider two columns X and Y, and the techniques can be easily extended to support one column or more than 2 columns. Without loss of generality, we assume that X is for x-axis and Y is for y-axis. Next we discuss how to transform X, Y to X', Y', by considering the two transformation operators (`GROUP BY` and `BIN`). We categorize the rules as follows.

I) *X is categorial:* we can only group X (cannot bin X). After generating the groups, we apply aggregation functions on Y for two cases. (i) If Y is numerical, we can apply an operation in $\texttt{AGG} = \{\texttt{AVG}, \texttt{SUM}, \texttt{CNT}\}$. (ii) If Y is not numerical, we can only apply `CNT`. Thus, we have two rules.
- $\mathbf{T}(X) = \texttt{Cat}, \mathbf{T}(\texttt{Y}) = \texttt{Num} \rightarrow \texttt{GROUP BY(X)}, \texttt{AGG(Y)}$.
- $\mathbf{T}(X) = \texttt{Cat}, \mathbf{T}(\texttt{Y}) \neq \texttt{Num} \rightarrow \texttt{GROUP BY(X)}, \texttt{CNT(Y)}$.

II) *X is numerical:* we can only bin X (cannot group X). After generating the buckets, we can apply aggregation functions on Y. (i) If Y is numerical, we can apply an operation in $\texttt{AGG} = \{\texttt{AVG}, \texttt{SUM}, \texttt{CNT}\}$. (ii) If Y is not numerical, we can only apply `CNT`. Thus we have two rules.
- $\mathbf{T}(X) = \texttt{Num}, \mathbf{T}(\texttt{Y}) = \texttt{Num} \rightarrow \texttt{BIN(X)}, \texttt{AGG(Y)}$.
- $\mathbf{T}(X) = \texttt{Num}, \mathbf{T}(\texttt{Y}) \neq \texttt{Num} \rightarrow \texttt{BIN(X)}, \texttt{CNT(Y)}$.

III) *X is temporal:* we can either group or bin X. After generating the groups or buckets, we can apply aggregation functions on Y. (i) If Y is numerical, we can apply an operation in $\texttt{AGG} = \{\texttt{AVG}, \texttt{SUM}, \texttt{CNT}\}$. (ii) If Y is not numerical, we can only apply `CNT`. Thus we have the following rules.
- $\mathbf{T}(X) = \texttt{Tem}, \mathbf{T}(\texttt{Y}) = \texttt{Num} \rightarrow \texttt{GROUP BY/BIN(X)}, \texttt{AGG(Y)}$.
- $\mathbf{T}(X) = \texttt{Tem}, \mathbf{T}(\texttt{Y}) \neq \texttt{Num} \rightarrow \texttt{GROUP BY/BIN(X)}, \texttt{CNT(Y)}$.

Example 8. Consider Table 1. If X = *carrier* (categorial) and Y = *passengers* (numerical), we can apply `GROUP BY`(*carrier*), `AVG`(*passengers*) and get Fig. 5b. If X = *scheduled* (temporal) and Y = *departure delay* (numerical), we can apply `BIN`(*scheduled*), `AVG`(*departure delay*) and get Fig. 1c.

2. Sorting Rules. Given two (transformed) columns, we can sort either X or Y. Intuitively, we sort numerical and temporal values in X but cannot sort categorical values. Note we can sort numerical values in Y;

otherwise it does not make sense. Thus we get the following rules.

- $\mathbf{T}(X) = \texttt{Num/Tem} \rightarrow \texttt{ORDER BY(X)}$.
- $\mathbf{T}(Y) = \texttt{Num} \rightarrow \texttt{ORDER BY(Y)}$.

Example 9. Based on Fig. 1c, we can sort *scheduled* (temporal column) and get a trend of average *departure delay*, which shows average *departure delay* fluctuates over time. It stands at the first relative high point $\sim$ 11:00, after which it starts to decline and rises again and reaches the peak $\sim$ 19:00.

3. Visualization Rules. For Y, it can be a numerical column but cannot be other types of columns.

I) If X is *categorical*, Y is *numerical*, we can only draw bar charts and pie charts.
II) If X is *numerical*, Y is *numerical*, we can draw the line charts and bar charts. Moreover, if X, Y have correlations, we can also draw scatter charts.
III) If X is *temporal*, Y is *numerical*, we draw line charts.

Thus we can get the following rules.

- $\mathbf{T}(X) = \texttt{Cat}, \mathbf{T}(\texttt{Y}) = \texttt{Num} \rightarrow \texttt{bar/pie}$.
- $\mathbf{T}(X) = \texttt{Num}, \mathbf{T}(\texttt{Y}) = \texttt{Num} \rightarrow \texttt{line/bar}$.
- $\mathbf{T}(X) = \texttt{Num}, \mathbf{T}(\texttt{Y}) = \texttt{Num}$, (X, Y) correlated $\rightarrow \texttt{scatter}$.
- $\mathbf{T}(X) = \texttt{Tem}, \mathbf{T}(\texttt{Y}) = \texttt{Num} \rightarrow \texttt{line}$.

Example 10. Fig. 5b is a meaningful bar chart, which consists of categorical column *carrier* as X and numerical column *passengers* as Y.

5.2 Rule-Based Visualization Selection

An Enumeration Algorithm. A straightforward algorithm enumerates every column pairs. (We need to consider both (X, Y) and (Y, X).) For each pair (X, Y), we enumerate every transformation rule. If the rule can be applied, we transform the data in the two columns into (X', Y'). Then we enumerate every sorting rule and transform it into (X'', Y''). Next, we try different visualization rules and draw the charts if the rule can be applied to (X'', Y'').

Based on these rules, we can get a set of visualization candidates. Next we use them to construct a graph and select top-k visualizations from the graph. However, this algorithm is rather expensive as it requires to first enumerate all candidates and then identify top-k ones from the graph. Next we propose optimization techniques.

A Progressive Method. We propose a progressive method to improve the performance of identifying top-k visualizations. The basic idea is that we do not generate all candidate visualizations, while progressively generate the candidates with the largest possibility to be in the top-k results.

Algorithm Overview. For each type of column, categorical, temporal, numerical, we keep a list of charts *w.r.t.* the column type, i.e., $\mathcal{L}_c$, $\mathcal{L}_t$, $\mathcal{L}_n$. We progressively generate the lists. For each list, we split it into different sublists based on the columns, we use $\mathcal{L}_c^X$ to denote the list of charts that take the categorical column X as x-axis. We can similarly define $\mathcal{L}_t$, $\mathcal{L}_n$ for temporal and numerical columns. Then we build a tree-like structure. The dummy root has three children $\mathcal{L}_c$, $\mathcal{L}_t$, $\mathcal{L}_n$. Each node $\mathcal{L}_c$ has several children, e.g., $\mathcal{L}_c^X$, for each categorical column X in the table. Next we use the tournament-like algorithm to select the best chart from leaf to root. For leaf nodes, we generate the best visualization in each leaf node *w.r.t.* the partial order. Then for each node $\mathcal{L}_c$, we select the best visualization from the visualizations of its children. Similarly from the root, we can select the best visualization from its children. If the best chart is selected from $\mathcal{L}_c^X$, we get the next best chart from the list and adjust the tournament. After we get k charts, it terminates.

Computing the Best Chart From $\mathcal{L}_c^X$ in the Leaf Node. For each list $\mathcal{L}_c^X$, we can only generate the bar chart and pie chart. We can get a list of charts based on each factor. Then we get the best one from these lists.

Computing the Best Chart From $\mathcal{L}_t^X$ in the Leaf Node. For each list $\mathcal{L}_t^X$, we only generate the scatter chart. We get a list of charts based on each factor and get the best one from these lists.

Computing the Best Chart From $\mathcal{L}_n^X$ in the Leaf Node. For each list $\mathcal{L}_n^X$, we can only generate the line chart and bar chart. We can get a list of charts based on each factor. Then we get the best one from these lists.

Computing the Best Chart From $\mathcal{L}_c/\mathcal{L}_t/\mathcal{L}_n$. We just need to select the best one from its children.

Computing the Best Chart From the Root. We compare different charts from its children and select the best one.

Based on the tournament we can generate the top-k charts without generating all the candidate charts.

Optimizations. Now, we propose several optimization techniques to improve the performance.

First, for each column X, when grouping and binning the column, we compute the `AGG` values on other columns together and avoid binning/grouping multiple times.

1) For each categorical/temporal column, we group the tuples in D and compute the `CNT` value; for each numerical column, we compute the `AVG` and `SUM` values in each group. Next we visualize the data based on the visualization rules.
2) For each temporal column, we bin the tuples in D, and compute the `CNT` value; for each numerical column, we compute the `AVG` and `SUM` values in each bin. Next we visualize the data based on the visualization rules.
3) For each numerical column, we bin the tuples in D, and compute the `CNT` value; for each numerical column, we compute the `AVG` and `SUM` values in each group. Next we visualize the data based on the visualization rules.

Second, we do not generate the groups of a column if there have k charts in $\mathcal{L}_c$ better than any chart in this column. Third, we postpone many operations after selecting the top-k charts, e.g., sorting, `AVG` operations. Thus we avoid many unnecessary operations that are not in top-k.

5.3 Rule Generation and Completeness

Below, we will discuss the "completeness" of rules introduced in Section 5.1, in terms of that they cover all cases that a visualization can potentially be meaningful (or good).

Transformation Rule Generation and Completeness. For transformation rule, we only consider categorical, numerical, and temporal columns. For categorical column, we can only apply group operations on it and apply aggregation on other columns. For numerical and temporal columns, we can only apply bin operations on it and apply aggregation on other columns. We can see that our rules consider all the possible cases and the transformation rules are complete.

Sorting Rule Generation and Completeness. It is trivial to generate sorting rules because we can only sort the numerical and temporal values on x-axis and numerical values on y-axis. We can see that our rules consider all the possible cases and the sorting rules are complete.

Visualization Rule Generation and Completeness. We only need to consider categorical, numerical, and temporal columns. We can only put the numerical columns on y-axis, and put categorical, numerical, and temporal columns on x-axis. For each case, there are four possible charts. Our rules consider all cases and the visualization rules are complete.

6 Diversified Visualizations Selection

The visualization selection method in Section 4 may select "*similar*" visualizations but cannot provide "*diversified*" visualizations. For example, the method may return many bar charts with high scores but only a few line charts. Naturally, the user wants these results to be *diversified* [4]. To address this issue, we propose a diversified visualizations selection method. Specifically, we treat the top-k visualizations selection problem as a bi-criteria optimization problem, which considers both the visualization score and diversity. Next we define the diversity in our problem first and then introduce our diversified top-k visualizations selection algorithm.

We measure the diversity of two visualizations from five aspects, i.e., visualization types, x-axis, y-axis, group/bin operations, aggregate functions. Thus the feature vector of a visualization v_i is denoted as $\mathbf{x}_i = [\mathbf{x}_i^1, \mathbf{x}_i^2, \mathbf{x}_i^3, \mathbf{x}_i^4, \mathbf{x}_i^5]$. More specifically, $\mathbf{x}_i^1$ is a one-hot vector, which encodes the four visualization types: *bar*, *pie*, *line*, and *scatter*. For example, $\mathbf{x}_i^1 = [1, 0, 0, 0]$ represents a bar chart; $\mathbf{x}_i^2$ (resp. $\mathbf{x}_i^3$) is also a one-hot vector with length m, denoting which column is used by the x-axis (resp. y-axis) of the visualization v, where m is the number of columns of the input table; $\mathbf{x}_i^4$ is a one-hot vector with length 3, which denotes the operations `GROUP BY`, `BIN` or NA; $\mathbf{x}_i^5$ is a one-hot vector with length 4, which denotes aggregation functions `SUM`, `AVG`, `CNT`, or NA. Next, we concatenate $\mathbf{x}_i = [\mathbf{x}_i^1, \mathbf{x}_i^2, \mathbf{x}_i^3, \mathbf{x}_i^4, \mathbf{x}_i^5]$ to a vector $\mathbf{x}_i$. Then given two visualizations v_i and v_j, we can measure the diversity between v_i and v_j by well-known similarity function, e.g., cosine similarity, i.e., $D(v_i, v_j) = 1 - \mathsf{Cosine}(\mathbf{x}_i, \mathbf{x}_j) = 1 - \frac{\vec{\mathbf{x_i}}\vec{\mathbf{x_j}}}{||\vec{\mathbf{x_i}}|| \, ||\vec{\mathbf{x_j}}||}$.

Next, we formally define our diversified top-k visualizations selection problem as below.

Definition 3 [Diversified Top-k Visualizations Selection]. *Given a list of visualizations V with size n, the diversified top-k visualizations selection problem aims to compute a list of visualizations $R \subseteq V$ such that*

$$R = \mathop{\arg\max}_{R \subseteq V, |R| = k} \mathcal{F}(V), \tag{10}$$

where $\mathcal{F}(V) = (1 - \lambda) \sum_{v_i \in V} S(v_i) + \frac{2\lambda}{k-1} \sum_{v_i, v_j \in V} D(v_i, v_j)$.

$\lambda \in (0, 1]$ is a parameter controlling the trade-off between the *visualization score* and *diversity*, which can be set by the user. The intuition behind this definition is that we aim to maximize $\mathcal{F}(V)$ so that we can derive a list of visualizations with relatively high visualization score as well as high diversity. We observe that there are k elements in the visualization score sum (i.e., $\sum_{v_i \in V} S(v_i)$) and $\frac{k(k-1)}{2}$ numbers in the *diversity* part. Therefore, we scale down the diversity part by $\frac{2\lambda}{k-1}$.

Unfortunately, the diversified top-k visualizations selection problem is NP-hard as proved below.

Theorem 1. *The diversified top-k visualizations selection problem is NP-hard.*

Proof. A special case of our problem is when $\lambda = 1$, it is equivalent to the *max-sum dispersion* problem [14], which is NP-hard. Therefore, the diversified top-k visualizations selection problem includes the *max-sum dispersion* problem and thus our problem is NP-hard. □

Algorithm 2. DiversifiedTopKVisSelectio

Input: visualizations list $V = [v_1, v_2, \ldots, v_n]$, k, λ;
Output: diversified top-k visualizations list R;
1: $R.append(v_1)$;
2: **for** ***each*** v_i *in* V **do**
3: *Added* = *True*;
4: **for each**v_j *in* R **do**
5: **if** $D(v_i, v_j) < \lambda$ **then**
6: *Added* = *False*;
7: break;
8: **if** *Added* = *True* **then** $R.append(v_i)$;
9: **if** $|R| = k$ **then** break;
10: **return** R;

There exists a 2-approximation algorithm [15] to solve the *max-sum dispersion* problem. We can adapt this algorithm to our problem. The key idea is that given the visualization list V, it finds a pair of visualizations (v_i, v_j) with the maximum $\mathcal{F}(\{v_i, v_j\})$, adds them into the result list, removes them from V and repeats until k visualizations are derived. However, the time complexity of this algorithm is $\mathcal{O}(kn^2)$ because it costs $\mathcal{O}(n^2)$ to compute the pairwise diversity scores and $\mathcal{O}(k)$ to find the results. The 2-approximation algorithm has two drawbacks: (i) it incrementally builds the result list R by selecting a pair of visualizations (v_i, v_j) with maximum $\mathcal{F}(\{v_i, v_j\})$ in each iteration, but fails to consider the diversity between current visualization pair (v_i, v_j) and other pairs already in the result list R; and (ii) it results in high computational cost and cannot meet DeepEye's interactive speeds requirement.

Therefore, we propose a heuristic algorithm to find diversified top-k visualizations effectively and efficiently, as shown in Algorithm 2. The key idea is that it first selects a visualization with the highest score, greedily adds the next one with the highest score if and only if it has a high diversity score with every visualization in the result list.

The pseudo code is shown in Algorithm 2. It takes as input the visualization ranking list V obtained by partial order-based approach, the number of visualizations k to be selected, and the diversity threshold λ, where a larger λ value indicates a higher diversity. The input visualization list V is sorted by visualization scores in descending order. It first adds the visualization with the highest score v_1 from V into the result list R (Line 1). Next, we iterate each visualization $v_i \in V$ in

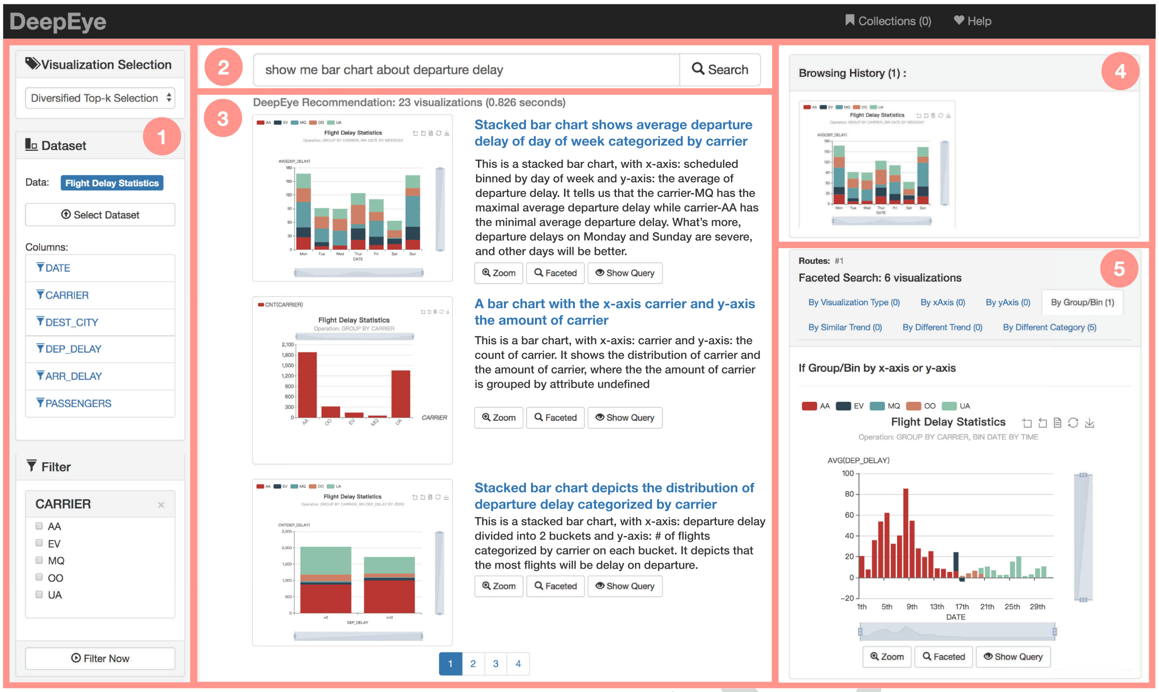


Fig. 7. DEEPEYE Screenshot. Part-❶ is responsible for dataset specification. The user can upload or select a dataset for visualization. It also supports filtering data at the "Filter" Panel. Panel-❷ is a keyword search box that the user can input the keyword query. DEEPEYE will suggest good visualizations relevant to the keyword query. Panel-❸ shows visualizations recommended by the system. For each visualization, DEEPEYE provides explanations to help the user better understand the result. The user can zoom in by clicking "Zoom" button for more details. The user can pick a visualization to do faceted navigation by clicking the "Faceted" button. The results of faceted navigation are shown in part-❺.

descending order, and check whether the diversity scores between v_i and any visualization v_j in R is smaller than the threshold (Lines 4–7). If so, we just drop it; otherwise we add v_i into R (Line 8).

The algorithm iterates the above steps until k visualizations are selected or all visualizations in V are visited. The time complexity of our algorithm is $\mathcal{O}(kn)$, which is much more efficient compared with the approximate algorithm.

7 DEEPEYE SYSTEM

DEEPEYE contains two main modules, a front-end user interface, which handles the interaction with the user, and a back-end service, which recommends visualizations based on the front-end requests. The front-end is a web-based user interface (Fig. 7). It allows users to upload their dataset to do data visualization. Then the back-end of DEEPEYE can automatically recommend meaningful visualizations to provide the user with "self-driving data visualization". Besides, the user can input a keyword query to obtain visualizations relevant to the query and explore the space of visualization results by faceted navigation.

(1) User Experience With Self-Driving Data Visualization. In self-driving data visualization, the user uploads her dataset (e.g., a CSV file) to DEEPEYE without any other operations. Then DEEPEYE will recommend meaningful visualizations to the user efficiently (see Fig. 7-❸). In this case, users can simply browse the recommendation list to pick their target visualizations.

(2) User Experience With Steerable Self-Driving Data Visualization. The user can "steer" by the following modules.

→ *Keyword Search.* Instead of guessing the users intent, DEEPEYE takes a keyword query as input and suggests some visualizations relevant to the keywords. In a nutshell, given a keyword query, DEEPEYE tries to find those visualizations whose corresponding queries match the query keywords. We first enumerate all possible visualization queries, called *candidate queries*, based on decision rules in Section 5.1. Next, we tokenize the keyword query into a set of words/phrases by n-gram techniques. For each word/phrase, we identity their mapping types, i.e., reserved keywords in our query (e.g., group by), table name (e.g., flight delay), column name (e.g., departure delay) and values in columns. In this way, we can get all relevant *candidate queries* and rank them by traditional ranking algorithms. Note that we can utilize WordNet [16], [17], [18] to identify synonyms and string similarity functions (e.g., Edit distance) to tolerate spelling mistake, which can further improve the quality of mapping. Thus we can select a set of *candidate queries* possibly relevant to the user's intent. Each *candidate query* can generate a visualization. Next, we return visualizations based on the generated *candidate queries*.

Now the user can query the visualizations by keyword search. For example, given a query "*Show me bar chart about departure delay*" (see Fig. 7–❷), DEEPEYE will recommend bar charts relevant to the attribute *departure delay*. That is, DEEPEYE will fix one attribute *departure delay* and discover other attributes that when being combined with attribute *departure delay* using an appropriate type of visualization, will produce good visualizations.

→ *Faceted Navigation.* When the user browses the visualizations list generated either by keyword search or *self-driving* mode, she/he can pick one good visualization, and do further faceted navigation to find other "interesting" visualizations by facets among multiple visualizations. Different from traditional faceted navigation, e.g., faceted

TABLE 5
Statistics of Experimental Datasets

#-Tuples			#-Columns		
Max	Min	Avg	Max Temporal/Categorical/Numerical/All	Min	Avg
99527	3	3381	2/12/21/25	0/0/1/2	1/2/5/7

TABLE 6
Ten Test Datasets

No.	Datasets	#-Tuples	#-Columns	#-Charts
X1	*Hollywood's Stories*	75	8	48
X2	*Foreign Visitor Arrivals*	172	4	10
X3	*McDonald's Menu*	263	23	275
X4	*Happiness Rank*	316	12	123
X5	*ZHVI Summary*	1,749	13	36
X6	*NFL Player Statistics*	4,626	25	209
X7	*Airbnb Summary*	6,001	9	42
X8	*Top Baby Names in US*	22,037	6	17
X9	*Adult*	32,561	14	103
X10	*Flight Delay*	99,527	6	44

navigation in E-commerce websites, where the facets can be easily defined by the categories of items, it is not easy to define the facets for visualizations because there is no consensus about criteria of finding interesting visualizations. Therefore, we propose to define facets based on the clause of our visualization query. We support facets that are closely related to the current visualization technologies, such as similar visualizations, different (or deviated) visualizations, various visualization types, various x/y axis, different aggregate functions, or changing group/bin operations. Next, when a user selects a visualization, we can recommend some visualizations with the same facets. Suppose that the user selects the *first stacked bar chart* (Fig. 7–❸) by clicking the "*Faceted*" button, DEEPEYE will suggest appropriate facets and return visualizations to the user. We can see from Fig. 7–❺ that, the suggested facets for this selected visualization are visualization type, x-axis, y-axis, group/bin, similar trend, different trend, and different category. The first chart under the facet "By Group/Bin" is a stacked bar chart with different group/bin operation compared with the selected one. The stacked bar chart under facets "By Group/Bin" first groups by carrier and then bins x-axis into buckets by the day of the month. It depicts the distribution of average departure delay during the day of the month and the daily average departure delay for each carrier. Note that, the user can do further faceted navigation iteratively, which may get more meaningful visualizations. Fig. 7–❹ keeps track of the visualizations that the user already browsed.

→ *Interactive Refinement*. DEEPEYE supports popular interactions such as zoom in/zoom out by leveraging an interactive visualization library ECHARTS (http://echarts.baidu.com). DEEPEYE also generates natural language explanation for each visualization to help the user better understand the visualization and data based on a rule-based translation method [19], [20]. Besides, the user can check the visualization query and can also customize the visualization by modifying the visualization query, especially for expert users.

8 EXPERIMENTS

The key questions we answered in this evaluation are: (1) How does DEEPEYE work for real cases? (2) How well does DEEPEYE perform in visualization recognition? (3) Whether the visualization selection of DEEPEYE can well capture human perception? (4) How does keyword search component work in visualization tasks? (5) How efficient is DEEPEYE?

8.1 Experimental Setup

Datasets. We have collected 42 datasets from various domain such as real estate, social study, and transportation. The statistical information are given in Table 5: the number of tuples ranges from 3 to 99,527, with an average 3,381; the number of columns is from 2 to 25; the statistics of #-columns for temporal, categorical, numerical is also given. Note that we assume that the dataset is clean enough for visualization. For those dirty data, we can first employ data cleaning techniques [21], [22] to clean data errors and then for visualization in DEEPEYE.

Ground Truth. We have asked 100 students to label the dataset. (1) For each dataset, we enumerated all the possible candidate visualizations and asked them to label which are good/bad. (2) For good visualizations, we asked them to compare two visualizations which are better. Then we merged the results to get a total order [23]. We got 2520/30892 annotated good/bad charts, and 285,236 comparisons for visualization pairs. Note that if a table has n visualizations, there are $\frac{n(n-1)}{2}$ rankings for one table.

Training. We selected 32 datasets as training datasets and trained ML models based on the ground truth of 32 datasets. We tested on other 10 datasets – this can help justify whether the trained ML models can be generalized. These 10 tables are given in Table 6, which are selected to cover different domains, various number of tuples and columns. Note that the last column, #-charts, refers to good visualizations. We also conducted cross validation and got similar results.

Experimental Environment. All experiments were conducted on a MacBook Pro with 8 GB 2133 MHz RAM and 2.9 GHz Intel Core i5 CPU, running OS X Version 10.12.3.

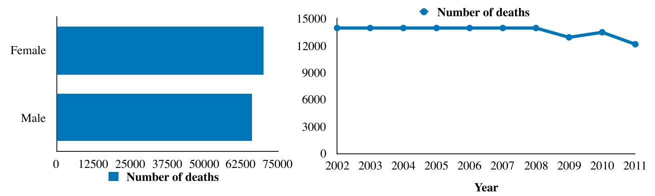
8.2 Experimental Results

Exp-(1): Coverage in Real Visualization Tasks. The most important item on nearly everybody's wish list is to see how DEEPEYE works for real visualization tasks. We collected 10 real-world visualization tasks in Table 7 (different from the above 42 datasets) with both datasets and visualizations. Each task is provided by senior users or domain experts on the internet.

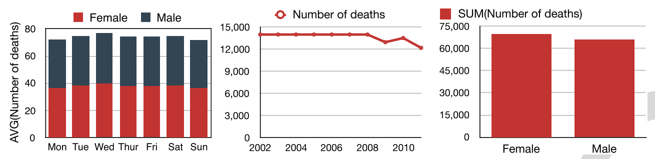
For example, T1.*Healthcare* is a visualization task on website (http://getdataseed.com) generated by a data analyst. The data analyst designed two visualizations to visually analyze T1.*Healthcare* dataset. The two visualizations are given in Fig. 8a. The bar chart depicts how many men and women died (more women died), while the line chart shows the number of deaths change over time. Next, we ran the dataset of T1 on DEEPEYE to verify whether DEEPEYE can recommend such good visualizations. We used partial order-based visualization selection method in this experiment. Fig. 8b is the top-3 results of running DEEPEYE on T1. This is the best case since all 2 visualizations used by the website

TABLE 7
Ten Visualization Tasks With Data and Visualizations

Visualization Tasks	Example Visualizations Source (URL)
T1.*Healthcare*	https://getdataseed.com/demo/
T2.*Flight Statistics*	https://www.transtats.bts.gov/airports.asp?pn=1
T3.*US Baby Names*	https://deepsense.io/us-baby-names-data-visualization/
T4.*Happy Country*	http://www.kenflerlage.com/2016/08/whats-happiest-country-in-world.html
T5.*Titanic Data*	https://public.tableau.com/profile/vaibhav.bhagat#!/vizhome/BIProject_4/Dashboard1
T6.*Avg. Food Price* T7.*China Economy*	http://data.stats.gov.cn/english/vchart.htm
T8.*Unemployment* T9.*Employment* T10.*Income&Wages*	https://www.maine.gov/labor/cwri/index.html



(a) 2 Visualizations of T1. *Healthcare*



(b) Top-3 visualizations suggested by DEEPEYE on T1 *Healthcare*

Fig. 8. Case study on visualization Task–T1.

TABLE 8
Coverage by DEEPEYE (The #-Visualizations Used in the Existing Visualization Tasks are Covered by Top-k Results in DEEPEYE)

Tasks	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10
#-Vis	2	4	5	5	8	27	35	6	4	4
Top-k	3	6	11	23	40	27	35	17	11	18

(data analyst) are automatically discovered by DEEPEYE in the top-3 visualizations. Besides, DEEPEYE also recommend other novel and interesting visualizations, e.g., the stacked bar chart in Fig. 8b, for the dataset of T1.

Note that traditionally, this will take hours for experienced data analysts who know the data very well to produce; now, you blink and it's done.

Applying DEEPEYE for other datasets are shown in Table 8. Take task T2 for instance, Table 8 shows that T2 has 4 practically used visualizations, which can be covered by top-6 results from DEEPEYE.

We have two main research findings from this group of experiment. (1) DEEPEYE can automatically discover visualizations needed in practice to tell compelling stories, which makes creating good visualizations a truly sexy task. (2) Sometimes the k visualizations needed to cover real cases is much larger than the #-real ones, e.g., it needs top-40 results to cover the 8 real cases in task T5. This is not bad at all since (i) users just browse few pages to find the ones they need; (ii) the other results not used by the real cases are not necessarily bad ones (some of them are

TABLE 9
Avg. Effectiveness (%): B (Bar), L (Line), P (Pie), S (Scatter) **A** (Avg)

	Precision			Recall			F-measure		
	Bayes	SVM	DT	Bayes	SVM	DT	Bayes	SVM	DT
B	84.30	86.90	93.20	84.10	86.40	93.00	84.20	86.65	93.10
L	93.20	96.50	99.50	90.80	96.40	99.50	91.98	96.45	99.50
P	83.40	90.60	94.70	82.60	90.60	94.70	83.00	90.60	94.70
S	84.30	86.90	93.10	84.10	86.40	92.90	84.20	86.65	93.00
A	86.30	90.23	**95.13**	85.40	89.95	**95.03**	85.85	90.09	**95.08**

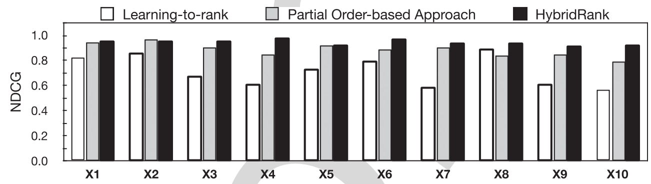


Fig. 9. Average effectiveness of visualization ranking & selection.

novel and interesting), for many cases the users may like them if they have seen them.

Exp-(2): Effectiveness of Visualization Recognition. Our main purpose in this group of experiment is to test (1) whether binary classifiers can well capture human perception for visualization recognition; and (2) which ML model best fits our studied problem?

We tested three popular ML models – Bayes, SVM and decision tree (DT). We used precision (P), recall (R) and F-measure (i.e., the harmonic mean of precision and recall).

Table 9 shows the effectiveness for bar (B), line (L), pie (P), and scatter (S) charts, which is the average of the 10 tested datasets. (**A**) shows the average results of four types of visualizations. We can see that the decision tree performs best and achieves averagely 95.08 percent F-measure–this justifies decision tree as a good choice for visualization recognition problem. The main reason is that the visualization recognition should follow the rules as discussed in Section 5.1 and decision tree could capture these rules well.

Exp-(3): Effectiveness of Visualization Selection. We used the normalized discounted cumulative gain (NDCG) [24] as the measure of ranking quality, which calculates the gain of a result based on its position in the result list and normalizes the score to $[0, 1]$ where 1 means perfect top-k results by comparing with the ground truth. We compared the NDCG values of learning-to-rank model, partial order-based approach, and HybridRank for 10 datasets X1-X10.

Fig. 9 reports the results. It shows clearly that partial order is better than learning-to-rank. The maximal NDCG of partial order is 0.97, and minimal NDCG of partial order is 0.81, while the maximal and minimal NDCG of learning-to-rank are 0.85 and 0.52, respectively. This is because the partial order ranked the visualization based on expert rules which captures the ranking features very well but learning-to-rank cannot learn these rules very well. HybridRank outperforms learning-to-rank and partial order-based visualization selection approach. For example, the average NDCG of HybridRank for 10 datasets is 0.94 and outperforms learning-to-rank and partial order method by 32.4 and 6.8 percent respectively.

Overall, the general observation is that HybridRank performs best and the partial order-based approach beats learning-to-rank for visualization selection.

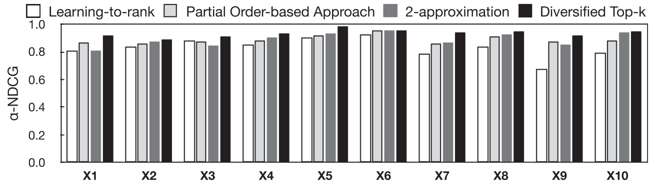


Fig. 10. Effectiveness of diversified top-k visualizations selection.

Next, we ran X1-X10 datasets on DeepEye to evaluate the effectiveness of our diversified top-k visualizations selection algorithm. We set $k = \frac{n}{2}$ for top-k and diversity parameter $\lambda = 0.5$. We compared our algorithm with learning-to-rank, partial order-based approach, and a 2-approximation diversified top-k visualizations selection algorithm. We utilize α-NDCG [25] as a metric to evaluate the diversity. α-NDCG is the variant of NDCG that balances relevance and diversity by rewarding diversity and penalizing redundant ones. We set $\alpha = 0.5$ as suggested by the literature [25].

The results for ten datasets are shown in Fig. 10. In general, we observe that learning-to-rank is the worst (i.e., 0.82 α-NDCG). The partial order is better than learning-to-rank but worse than other two diversified top-k selection algorithms among ten datasets. As expected, both our diversified top-k visualizations selection algorithm and the 2-approximation baseline work well. More concretely, our algorithm achieves averagely 0.93 α-NDCG and performs best. The 2-approximation baseline achieves averagely 0.90 α-NDCG. For a better understanding, we show running examples in Fig. 11, which shows the top-6 visualizations recommended by four methods. Both learning-to-rank and partial order-based approaches recommend visualizations that are individually to the interest of the user but with very similar trend or chart types. This is likely to make users feel boring when browsing those similar charts. In this case, the 2-approximation algorithm also suggests some homogeneous results such as the two scatter plots (the third one and the fourth one in Fig. 11c). Instead, as shown in Fig. 11d, the top-6 visualizations recommended by our algorithm are high diversity to each other, which can cover the four widely used types of visualizations.

Exp-(4): Usability of Keyword Search.

Our main purpose in this group of experiments is to test whether the keyword search component can save the interaction time to complete a visualization task.

First, we recruited 6 participants (1 female, 5 male) from the CS Department as real users to participate in this experiment. All participants have data analysis and visualization experience. Our experiment began with an introduction to the 10 datasets in Table 6 and a short tutorial about DeepEye. We considered two interaction methods: (1) Browse: participants only browse the visualizations list recommended by DeepEye to pick their desired visualization results. Note that, we used diversified visualizations selection method as default; and (2) Browse/Keyword Search: participants can browse the visualizations list and use the keyword search component alternately to find their desired visualization results. We asked each participant to perform a visualization task (i.e., picking their desired visualizations) on each dataset in two interaction methods respectively. We recorded the interaction time of each visualization task. Hence, there are 60 interaction time records for each type of interaction method.

We used box plots to concisely visualize the distribution of the interaction time of each interaction method. The middle line represents the median value of the records while the box boundaries correspond to the 25th and 75th percentiles. The top and bottom whiskers are set to denote the 95th and 5th percentiles respectively. As shown in Fig. 13, we can see that most of the visualization tasks can be completed in $65-110$ seconds under the Browse method. If we allow participants to use the keyword search component, the interaction time significantly reduces to $30-50$ seconds, which indicates that participants using the Browse/Keyword Search complete visualization tasks are much faster than them using the Browse method. The experimental results show that completing a visualization task using the keyword search component is more effective.

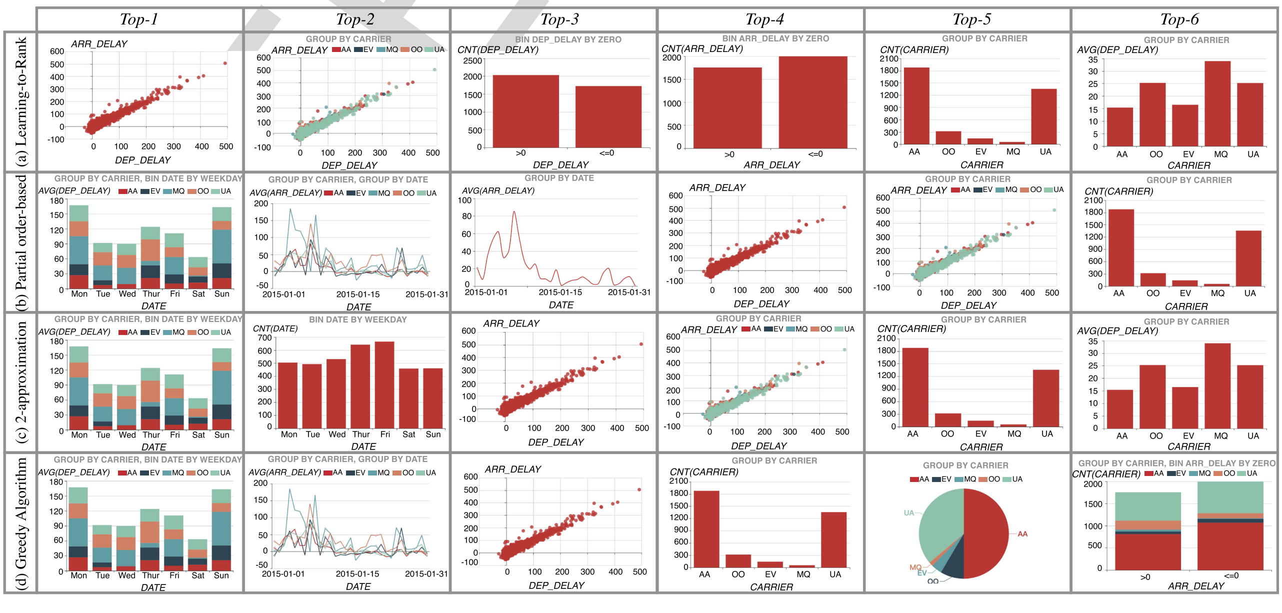


Fig. 11. Top-6 visualizations recommended by DeepEye.

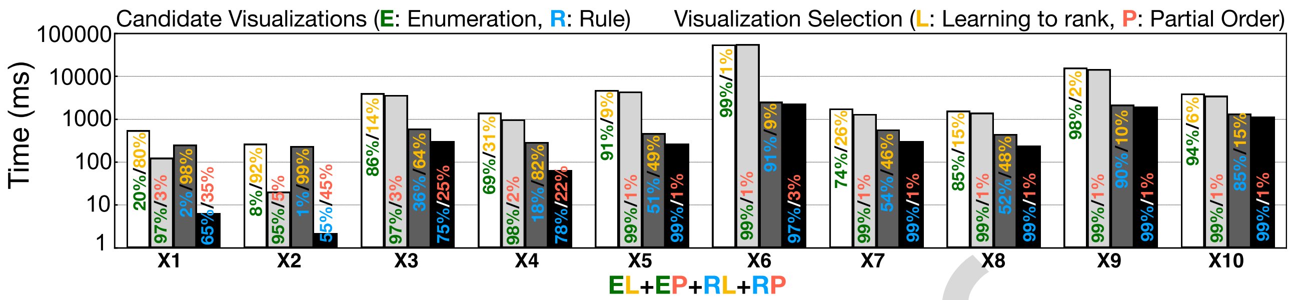


Fig. 12. Efficiency of DeepEye.

Exp-(5): Efficiency–Tell the Stories of Your Data in Seconds! We first compared the efficiency of our greedy diversified top-k visualizations selection algorithm with the 2-approximation algorithm (i.e., the baseline). We set $k = n$ (n is the total number of visualizations can be selected) and varied the #-visualizations. We repeated all experiments ten times to compute the average results. Fig. 14 reports the results. We can see that our algorithm is more efficient than the baseline, especially when the number of visualizations become larger. More concretely, for the dataset X3, the baseline takes 2239.20 ms to rank 275 visualizations, while our greedy algorithm only takes 2.34 ms.

We have also tested the efficiency of DeepEye on ten datasets X1–X10. Each dataset is associated with 4 bars that measure the end-to-end running time from a given dataset to visualization selection. The time of each bar consists of two parts: (i) generate all candidate visualization without/with (i.e., E/R) using our transformation/sorting/visualization rules; and (ii) visualization selection using learning-to-rank/partial order-based solutions. We annotate the percentage (%) of these two parts in each bar, e.g., the first bar means that it needs 550 ms, where visualization enumeration (E) takes 20 percent time and visualization selection using learning to rank (L) takes 80 percent.

Fig. 12 tells us the followings: (1) using the rules (Section 5.1) can effectively reduce the running time, i.e., RL (resp. RP) runs always faster than EL (resp. EP) since it avoids generating many bad visualizations, as expected; (2) partial order-based approach runs faster than learning to rank model, i.e., EP (resp. RP) runs always faster than EL (resp. RL), because partial order can efficiently prune the bad ones while learning to rank must evaluate every visualizations; (3) DeepEye can run to complete in seconds for datasets with reasonable size. Note that the performance will be further boosted by DBMSs (e.g., the database-based optimizations in SeeDB [2] and DeVIL [26]) or approximate query process technique [27].

9 Related Work

Visualization Recommendation. There has been work on recommending visualizations, such as SeeDB [2], Profiler [28], and Voyager [29]. SeeDB quantifies an "interesting" visualization as the one that is largely deviated from a user given reference, which is similar to find an outlier. Profiler is similar to SeeDB, which finds anomalies as candidate recommendations. Voyager suggests visualizations based on statistical properties of all visualizations.

Existing methods mainly use statistical properties (e.g., outliers) for visulization recommendations. Different from them, (1) DeepEye tries to capture the human perception by understanding existing examples using ML-based techniques; and (2) DeepEye can accept keyword search to do recommendation instead of guessing the user's preference.

Interactive Data Visualization Systems. DeVIL [26] employs a SQL-like language to support interactive visualization. zenvisage [3] tries to find other interesting data when the users provide their desired trends or patterns. Lyra [30] is an interactive environment that enables custom visualization design without writing code. VisClean [21] allows users to progressively improve the visualization quality by interactively cleaning data errors. DataTone [31] provides a natural language interface for visual analysis. It accepts natural language as input and iteratively interacts with the user to produce one visualization.

DeepEye allows the user to specify their intent by keywords and recommends a list of visualizations relevant to the keywords in one-shot. Besides, the user can further explore via faceted navigation.

Data Visualization Languages. There have been several works on defining visualization languages. ggplot [32] is a programming interface for data visualization. ZQL [3] borrows the idea of Query-by-Example (QBE) that has a tabular structure. Vega (https://vega.github.io/vega/) is a visualization grammar in a JSON format. VizQL [33] is a visual query

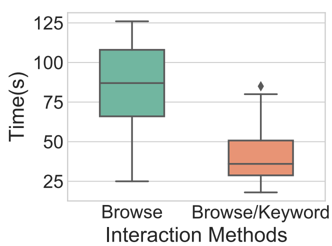


Fig. 13. User study results.

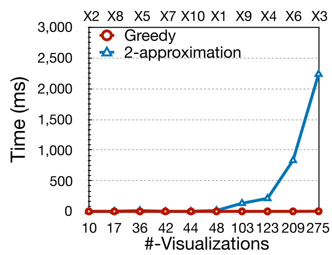


Fig. 14. Efficiency.

language that translates drag-and-drop actions into data queries and then expresses data visually.

Our proposed language is a subset, but shares many features with the others. Our purpose to define a simple language is just to make our discussion easier.

10 Conclusion

We have presented DeepEye, a novel self-driving data visualization system. We leveraged machine learning techniques as black-boxes and expert specified rules, to solve three challenging problems faced by DeepEye, namely, visualization recognition, visualization ranking, and visualization selection. We also study the problem of how to compute diversified top-k visualizations. In order to better capture a user's query intent, we further extend DeepEye to be easily steerable, by providing keyword search and faceted navigation. We have demonstrated its effectiveness and easy-to-use by using real-world datasets and use cases. Y. Luo and X. Qin are contributed equally to this research.

Acknowledgments

This work was supported by NSF of China (61925205, 61632016, 61521002), Huawei, TAL education, Tsinghua University Initiative Scientific Research Program, and Beijing National Research Center for Information Science and Technology (BNRist).

References

[1] X. Qin, Y. Luo, N. Tang, and G. Li, "Making data visualization more efficient and effective: A survey," *The VLDB J.*, vol. 29, pp. 93–117, 2019.
[2] M. Vartak *et al.*, "SEEDB: Efficient data-driven visualization recommendations to support visual analytics," *Proc. VLDB Endowment*, vol. 8, pp. 2182–2193, 2015.
[3] T. Siddiqui, *et al.*, "Effortless data exploration with zenvisage: An expressive and interactive visual analytics system," *Proc. VLDB Endowment*, vol. 10, pp. 457–468, 2016.
[4] M. Vartak *et al.*, "Towards visualization recommendation systems," *ACM SIGMOD Rec.*, vol. 45, pp. 34–39, 2017.
[5] C. Binnig, L. D. Stefani, and T. Kraska *et al.*, "Toward sustainable insights, or why polygamy is bad for you," in *Proc. 8th Biennial Conf. Innovative Data Syst. Res.*, 2017.
[6] Y. Luo, X. Qin, N. Tang, and G. Li, "DeepEye: Towards automatic data visualization," in *Proc. IEEE 34th Int. Conf. Data Eng.*, 2018, pp. 101–112.
[7] L. Grammel, M. Tory, and M.-A. Storey, "How information visualization novices construct visualizations," *IEEE Trans. Vis. Comput. Graphics*, vol. 16, no. 6, pp. 943–952, Nov./Dec. 2010.
[8] C. J. C. Burges *et al.*, "Learning to rank using gradient descent," in *Proc. 22nd Int. Conf. Mach. Learn.*, 2005, pp. 89–96.
[9] C. J. C. Burges, K. M. Svore, Q. Wu, and J. Gao, "Ranking, boosting, and model adaptation," Microsoft Research, Redmond, WA, Tech. Rep. MSR-TR-2008–109, 2008.
[10] J. D. Mackinlay, P. Hanrahan, and C. Stolte, "Show me: Automatic presentation for visual analysis," *IEEE Trans. Vis. Comput. Graphics*, vol. 13, no. 6, pp. 1137–1144, Nov./Dec. 2007.
[11] J. D. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Trans. Graph.*, vol. 5, pp. 110–141, 1986.
[12] W. S. Cleveland, *et al.*, "Graphical perception: Theory, experimentation, and application to the development of graphical methods," *J. Amer. Statist. Assoc.*, vol. 79, pp. 531–554, 1984.
[13] M. de Berg, O. Cheong, M. J. van Kreveld, and M. H. Overmars, *Computational Geometry: Algorithms and Applications*. Berlin, Germany: Springer, 2008.
[14] M. J. Kuby, "Programming models for facility dispersion: The p-dispersion d maxisum dispersion problems," *Geogr. Anal.*, vol. 19, pp. 315–329, 1987.
[15] R. Hassin, S. Rubinstein, and A. Tamir, "Approximation algorithms for maximum dispersion," *Operation Res. Lett.*, vol. 21, pp. 133–137, 1997.
[16] G. A. Miller, *WordNet: An Electronic Lexical Database*. Cambridge, MA, USA: MIT Press, 1998.
[17] B. Li *et al.*, "Scaling Word2Vec on big corpus," *Data Sci. Eng.*, vol. 4, pp. 157–175, 2019.
[18] Q. Zhu, X. Ma, and X. Li, "Statistical learning for semantic parsing: A survey," *Big Data Mining Analytics*, vol. 2, no. 4, pp. 217–239, Dec. 2019.
[19] Y. Luo *et al.*, "DeepEye: Creating good data visualizations by keyword search," in *Proc. Int. Conf. Manage. Data*, 2018, pp. 1733–1736.
[20] X. Qin, Y. Luo, N. Tang, and G. Li, "DeepEye: Visualizing your data by keyword search," in *Proc. 21st Int. Conf. Extending Database Technol.*, 2018, pp. 441–444.
[21] Y. Luo, C. Chai, X. Qin, N. Tang, and G. Li, "Interactive cleaning for progressive visualization through composite questions," in *Proc. IEEE Int. Conf. Data Eng.*, 2020.
[22] M. Li, H. Wang, and J. Li, "Mining conditional functional dependency rules on big data," *Big Data Mining Analytics*, vol. 3, no. 1, pp. 68–84, Mar. 2020.
[23] X. Zhang, G. Li, and J. Feng, "Crowdsourced top-k algorithms: An experimental evaluation," *Proc. VLDB Endowment*, vol. 9, pp. 612–623, 2016.
[24] H. Valizadegan, R. Jin, R. Zhang, and J. Mao, "Learning to rank by optimizing NDCG measure," in *Proc. 22nd Int. Conf. Neural Inf. Process. Syst.*, 2009, pp. 1883–1891.
[25] C. L. A. Clarke *et al.*, "Novelty and diversity in information retrieval evaluation," in *Proc. 31st Annu. Int. ACM SIGIR Conf. Res. Development Inf. Retrieval*, 2008, pp. 659–666.
[26] E. Wu *et al.*, "Combining design and performance in a data visualization management system," in *Proc. 8th Biennial Conf. Innovative Data Syst. Res.*, 2017.
[27] K. Li and G. Li, "Approximate query processing: What is new and where to go?" *Data Sci. Eng.*, vol. 3, pp. 379–397, 2018.
[28] S. Kandel *et al.*, "Profiler: Integrated statistical analysis and visualization for data quality assessment," in *Proc. Int. Work. Conf. Adv. Vis. Interfaces*, 2012, pp. 547–554.
[29] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer, "Voyager: Exploratory analysis via faceted browsing of visualization recommendations," *IEEE Trans. Vis. Comput. Graphics*, vol. 22, no. 1, pp. 649–658, Jan. 2016.
[30] A. Satyanarayan and J. Heer, "Lyra: An interactive visualization design environment," *Comput. Graph. Forum*, vol. 33, pp. 351–360, 2014.
[31] T. Gao *et al.*, "DataTone: Managing ambiguity in natural language interfaces for data visualization," in *Proc. 28th Annu. ACM Symp. User Interface Softw. Technol.* 2015, pp. 489–500.
[32] H. Wickham, "ggplot2 - elegant graphics for data analysis," Springer, 2009, doi: 10.1007/978-0-387-98141-3.
[33] P. Hanrahan, "VizQL: A language for query, analysis and visualization," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2006, Art. no. 721.

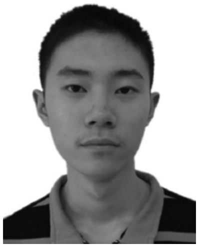
Yuyu Luo received the bachelor's degree in software engineering from the University of Electronic Science and Technology of China, China, in 2018. He is currently working toward the master's degree in the Department of Computer Science, Tsinghua University, Beijing, China. His research interests include data cleaning and data visualization.

Xuedi Qin received the bachelor's degree in computer science and technology from the Harbin Institute of Technology, China, in 2017. She is currently working toward the PhD degree in the Department of Computer Science, Tsinghua University, Beijing, China. Her research interests include data visualization and data exploration.

Chengliang Chai received the bachelor's degree in computer science and technology from the Harbin Institute of Technology, China, in 2015. He is currently working toward the PhD degree in the Department of Computer Science, Tsinghua University, Beijing, China. His research interests include crowdsourcing data management and data mining.

Nan Tang received the PhD degree from The Chinese University of Hong Kong, Hong Kong, in 2007. He is a senior scientist at QCRI, Qatar. He has worked as a research staff member at CWI, The Netherlands, from 2008 to 2010. He was a research fellow at the University of Edinburgh, Scotland, from 2010 to 2012. His current research interests include data curation and data streams.

Guoliang Li received the PhD degree in computer science from Tsinghua University, China, in 2009. He is currently working as a professor with the Department of Computer Science, Tsinghua University, Beijing, China. His research interests mainly include data cleaning and integration, spatial databases, and crowdsourcing.



Wenbo Li is currently working toward the undergraduate degree in the Department of Computer Science, Tsinghua University, Beijing, China. His research interest include data visualization.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.