Provenance-based Recommendations for Visual Data Exploration

Houssem Ben Lahmar Melanie Herschel

IPVS - University of Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany

{houssem.ben-lahmar | melanie.herschel}@ipvs.uni-stuttgart.de

Abstract

Visual data exploration allows users to analyze datasets based on visualizations of interesting data characteristics, to possibly discover interesting information about the data that users are a priori unaware of. In this context, both recommendations of queries selecting the data to be visualized and recommendations of visualizations that highlight interesting data characteristics support users in visual data exploration. So far, these two types of recommendations have been mostly considered in isolation of one another.

We present a recommendation approach for visual data exploration that unifies query recommendation and visualization recommendation. The recommendations rely on two types of provenance, i.e., data provenance (aka lineage) and evolution provenance that tracks users' interactions with a data exploration system. This paper presents the provenance data model as well as the overall system architecture. We then provide details on our provenance-based recommendation algorithms. A preliminary experimental evaluation showcases the applicability of our solution in practice.

Keywords Provenance application, visual data exploration

1. Introduction

Data exploration (Idreos et al. 2015) supports users in inspecting the contents of a given dataset, to reveal interesting and possibly unexpected information hidden in the data. The rationale behind data exploration is that users do not know what valuable information they seek, which requires an investigative way of navigating through (portions of) the data and analyzing the selected data. The analysis can be supported by offering a suited visualization of the underlying data to the user, who can then perform visual data exploration. Typically, users explore multiple portions of the data, selected, joined, aggregated, or visualized in different ways, rendering visual data exploration an interactive process.

TaPP 2017, June 22-23, 2017, Seattle, Washington.

Clearly, manual data exploration is a tedious and time consuming task. Therefore, automated methods to recommend either queries to select datasets to be explored or visualizations for a given datasets have recently been proposed (Drosou and Pitoura 2013; Wongsuphasawat et al. 2016; Vartak et al. 2015; Milo and Somech 2016).

However, these methods consider query recommendation and visualization recommendation separately from one another. This paper presents how to perform these recommendations in an integrated way. The fabric linking these two types of recommendations consists of provenance meta-data about the evolution of the visual data exploration process. In addition, query recommendation leverages data-provenance. Overall, to the best of our knowledge, this paper is the first to consider provenance meta-data to compute query and visualization recommendations for visual data exploration.

Contributions. This paper focuses on recommendations for visual data exploration of data stored in a relational data warehouse. Our contributions are the following:

- We introduce a data model for evolution provenance for visual data exploration applications and describe how it is maintained during visual data exploration.
- For SQL queries involving selection, projection, join, and aggregation, we present a query recommendation algorithm that leverages data provenance to guide users to different portions of the studied data sets. The recommended queries follow typical data warehouse operations such as drill-down, roll-up, or slice.
- We outline a method that uses evolution provenance to recommend visualizations for a query result, once a user has selected a (recommended) query. The rationale behind these recommendations is to provide visualizations that render semantically close information in a similar way. This allows users to more easily recognize information encountered earlier, potentially increasing the understandability or efficiency of visual data exploration.

Structure. Sec. 2 first provides an overview of our visual data exploration system. We discuss evolution provenance in Sec. 3 before we describe provenance-based recommendations of queries and visualizations in Sec. 4. Sec. 5 briefly introduces our system implementation and presents a prelim-

inary evaluation of the system's performance. Related work is covered in Sec. 6 before we conclude in Sec. 7.

2. System overview

The left side of Fig. 1 depicts the general functionality of our visual data exploration system, called EVLIN (short for exploring visually with lineage). The different components of the system are named at the top. The right hand side of Fig. 1 provides an example of the system's behavior.

EVLIN supports interactive visual data exploration over a multidimensional relational dataset D, typically stored in a data warehouse with star- or snowflake-schema, denoted $schema(D)$, allowing users to trigger an exploration session with a seed exploration query Q (Step ① in Fig. 1).

Definition 2.1 (Exploration query) *Given a fact table T, the set of measures M described in T, the set of dimension tables A, and a set of aggregate functions F, an exploration query Q is an SQL query of the form*

SELECT $f(m), a$ *FROM* $rel(Q)$ *WHERE* $cond$ *GROUP BY* a

where $m \in M$, $a \in A$, and $f \in F$ such that f applies on the measure m, $rel(Q)$ refers to one or more relations in $schema(D)$, and $cond$ is a (conjunction of) predicates.

The *query processor* executes the exploration query Q over the dataset D and its result $Q(D)$ is input to *rendering recommendation* in Step ②. This component determines an adequate visualization for $Q(D)$ to be displayed to the user.

Example 2.2 The sample query shown in Fig. 1 asks for the sum of all sales for the period 2014 to 2016 per company name. Once the query is executed, assume we get the visualization shown in Fig. 1 as an example for Step ②. ■

This visualization allows users to interact with the query result. In particular, users can select a sub-result $r \in Q(D)$ that has attracted their attention. In Step ③, r is processed by the *data recommendation* component that identifies which information may be of interest to the user at the next exploration step. *Query reformulation* processes these in Step ④ to suggest reformulations of the initial query Q. A scored list of recommended queries for the next exploration step is visualized in form of an impact matrix, allowing the user to choose which of the suggested queries Q' to execute next (Step ⑤). At this point, the next exploration step of the exploration session starts, by setting Q to Q'.

The two recommendation components rely on the collection and processing of provenance information. While data recommendation mainly relies on data provenance (Cheney et al. 2009), the rendering recommendation takes into account meta-data about the derivation of a (recommended) query and visualization as an exploration session evolves, which we subsequently call evolution provenance. Details about our data model for evolution provenance are discussed next, before we cover the recommendation components.

3. Modeling Evolution Provenance

In our context, evolution provenance tracks the individual exploration steps that compose an exploration session in which users navigate through the data as discussed in the previous section. Given that exploration steps include user interactions, recommendations, and visualizations, the evolution provenance model described in this section comprises, for each exploration step, meta-data about the explored data and its associated visual encoding parameters. During an exploration session, the evolution provenance of each exploration step is maintained and linked based on recommendations that users decided to follow.

Scientific workflow engines (Callahan et al. 2006; Altintas et al. 2006) leverage a similar type of provenance. Indeed, they track the changes to a workflow during its development, tuning, or evolution. However, the meta-data collected as part of the evolution provenance in scientific workflow engines is essentially based on changes of the workflow specification.
Opposed to that, the evolution provenance for visual data exploration proposed here describes an actual run of an exploration session, divided into individual exploration steps.

Definition 3.1 (Exploration step) *An exploration step is defined as $X = \{Q, D, V\}$ where Q is the query whose result $Q(D)$ over database D is visualized with an interactive visualization described by V.*

Further below, we will discuss that V is defined as a query over a schema modelling exploration resources. However, before delving into these details, we first define how individual exploration steps combine to an exploration session.

Definition 3.2 (Exploration session) *An exploration session is defined as an ordered labeled graph $G_X(N, E)$ where N is the set of nodes and E a set of labeled edges. Each node $n \in N$ corresponds to an exploration step X and an edge $e = (n, n')$ represents the transition from one exploration step X to the next exploration step X' defined by following a recommended query Q' with associated recommended visualization V'. Consequently, edge e is labeled with the recommendation pair (Q', V').*

We now turn our attention to the definition encoding a visualization for an exploration step. Inspired by (Wu et al. 2017), we record meta-data about *exploration resources* in a relational database. The design of its schema draws on a set of visualization specifications such as Vega (Satyanarayan et al. 2017) and D3 (Bostock et al. 2011), which define a visualization using three fundamental components. (1) Graphical *marks* describe the graphical forms adopted to visually represent data, e.g., circles, rectangles, or points. (2) A given graphical form is further specified using a set of encoding *channels* that describe the visual mapping of each attribute. (3) Finally, the *scale* component maps data values to visual values, e.g., specification of position encodings for such data

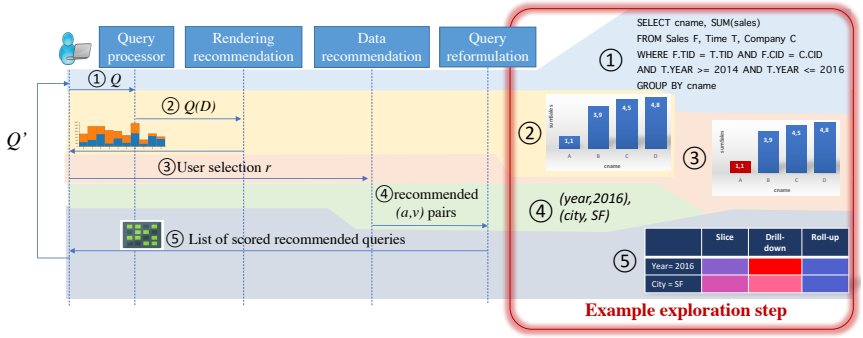

Figure 1: EVLIN system overview and example

Dataset(DSID, query)
Vis(VID, DSID → Dataset, width, height)
Scale(SID, VID → Vis, scType, fName, operator, range)
RenderedDS(RID, VID → Vis, selectedData)
Mark(MID,mType,RID → RenderedDS)
Channel(CID,MID → Mark, chType, chAtt, chVal)

Figure 2: Evolution provenance schema

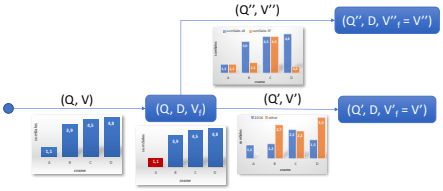

Figure 3: Exploration session graph

value. In addition to these main components, we further store information about data to render, or the width and the height of a view. As a visualization may render different fragments of the dataset $Q(D)$ differently (e.g., one bar in red while the other bars are blue), a visualization consists of several rendered data sets, each associated with graphical marks. The meta-data collected as part of an exploration resource will be used by our recommendation algorithms.

Definition 3.3 (Exploration resource) *An exploration resource V for an exploration step $X = \{Q, D, V\}$ is defined as the following query over the schema given in Fig. 2.*

$$V = \Pi_{Vis.*,RenderedDS.*,Scale.*,Mark.*,Channel.*}(\sigma_{query=Q}(Dataset) \bowtie Vis \bowtie RenderedDS \bowtie Scale \bowtie Mark \bowtie Channel)$$

Example 3.4 Let us consider the exploration step illustrated in Fig. 1. The initial query Q is the query labeled ①. The associated visualization is initially the bar chart labeled ②. Fig. 4 shows the corresponding tables describing the exploration resources at this stage. As the relations Dataset (a) and Vis (b) are self-explanatory, we focus the discussion on tables Scale (c), RenderedDS (d), Mark (e), and Channel (f). Scale specifies two scales for the visualization V1 in order to specify the geometric position of a rendered result. Given that all data is visualized in the same way (blue bars with labels on top), there is one rendered data set defined by query Q stored in RenderedDS. To render this data set, two marks are used: the bars (M1) and the text labels (M2) at the top of the bars. For each mark, several channels are used. The mark $M1$ uses two encoding channels $C1$ and $C2$ to specify the x-position of the bar (here, it is with respect to the company attribute) and its height on the y-axis (depending on the sumSales). A third channel $C3$ defines the blue fill color of rendered bars. Likewise, the text label mark $M2$ uses encoding channels $C4$ and $C5$ to specify the geometric position of a bar while $C6$ specifies that sales values are to be rendered as a text in the visualization at the top of a bar. Finally, $C7$ specifies that the text color should be white. ■

DSID	query
D1	Q

(a) Dataset

VID	DSID	width	height
V1	D1	500	700

(b) Vis

SID	VID	scType	fName	operator	range
xsc	V1	ordinal	cname	sort	width
ysc	V1	linear	sumSales	⊥	height

(c) Scale

RID	VID	selectedData
R1	V1	Q

(d)RenderedDS

MID	mType	RID
M1	bar	R1
M2	text	R1

(e) Mark

CID	MID	chType	chAtt	chVal
C1	M1	X	fName	company
C1	M1	X	fType	nominal
C2	M1	Y	fName	sumSales
C2	M1	Y	fType	numeric
C3	M1	fill	color	blue
C4	M2	X	fName	company
C4	M2	X	fType	nominal
C5	M2	Y	fName	sumSales
C5	M2	Y	fType	numeric
C6	M2	text	fName	sales
C6	M2	text	fType	numeric
C6	M2	text	align	top
C7	M2	fill	color	white

(f) Channel

Figure 4: Sample exploration resource representation

As is illustrated in Fig. 1, the visualization labeled ② is only the initial visualization associated to a query Q in the current exploration step. Indeed, as the user interacts with the system, the visualization changes, e.g., to the visualization shown in ③, where a selected bar is now highlighted. Consequently, we update the exploration resource associated to Q by sending SQL update statements to the underlying exploration resource database. This means that from the beginning of an exploration step to its end (i.e., when a recommended query is pursued), the visualization V of an exploration step may change. In our model of an exploration session, we record both the initial visualization V (as part of edge labels) as well as the final visualization V_f for each exploration step based on which users decided to move to another (recommended) query. The rationale behind this is that the V_f was deemed interesting for recommendation and

further exploration by the user while other (intermediate) visualizations were deemed less interesting and thus possibly less memorable. Consequently, intermediate visualizations are not considered by our recommendation algorithms.

Example 3.5 As a simple example of an exploration session, consider that in the example at Step ⑤ shown in Fig. 1, the user selects the drill-down query for year = 2016 (depicted as most relevant by the red color). This interaction moves from one exploration step to the next, with a recommended query Q' and associated visualization V'. The graph modelling the captured evolution provenance is depicted in Fig. 3. For the new exploration step, V'_f is set to V' initially. Based on the new visualized dataset, the user recognizes for instance that company A has no sales before 2016, possibly due to the fact that the company started to sell in 2016. Next, assume the user moves back one step to see how sales for company A are going in San Francisco (a slice query previously identified as relevant, given the impact matrix in Fig. 1). This creates another branch in our graph model. From this result, the user may infer that company A does well compared to its competitors in San Francisco. ■

4. Provenance-based recommendations

We now present our recommendation algorithm that assists users during an exploration session. The recommendations include both a recommended query and a corresponding visualization, which we discuss in sequence. Thus, the discussion focuses on the steps of data recommendation and rendering recommendation shown at the left hand side of Fig. 1.

Algorithm 1: Data recommendation algorithm

```
Input: Q, D, r, θ_L, θ_supp
Output: RMap: data recommendations in form of a mapping associating
        attribute labels a to value lists V_L
1  Lin(R) ← get data provenance of r wrt D ;
2  foreach a ∈ schema(Lin(r)) do
3      foreach v ∈ adom(a) do
4          f_Lin ← compute f_{a,v}(Lin(r)) using Equ. 4.1;
5          if f_Lin ≥ θ_L then
6              f_D ← compute f_{a,v}(D) using Equ. 4.1;
7              support ← supp_{a,v}(r) as defined by Equ. 4.2 ;
8              if support ≥ θ_supp then
9                  RMap ← mapInsert(RMap, (a, v)) ;
10 forall the a_i, a_j ∈ keys(RMap) × keys(RMap), i ≠ j do
11     if Functional dependency a_i → a_j holds then
12         mapRemove(RMap, a_j);
13 Return RMap;
```

4.1 Data recommendation

Our data recommendation algorithm exploits data provenance to recommend attributes and values of these attributes that may raise user interest. Alg. 1. shows its pseudocode.

The algorithm takes as input query Q, dataset D, a sub-result of interest $r \in Q(D)$ selected by a user, and two threshold values θ_L and θ_{supp}. In line 1, we first determine the data provenance of result r, denoted as $Lin(r)$. We compute data provenance using the Perm provenance management system. This provenance corresponds to all tuples in D that have contributed to producing the result tuple r.

Lines 2–9 perform the provenance-based identification of relevant data by iterating over attribute-value pairs (a, v), where a is an attribute in the schema of the provenance $Lin(r)$ and v is a value in the active domain of a, denoted $adom(a)$. Relevant attribute-value pairs (a, v) are pairs that appear frequently enough in the provenance while this frequency significantly differs from the appearance frequency of (a, v) in the whole database D.

More formally, we compute the appearance frequency for each value v of each attribute a in $Lin(r)$ as

$$f_{a,v}(Lin(r)) = \frac{|\{t_i | t_i \in Lin(r) \wedge t_i.a = v\}|}{|Lin(r)|} \qquad (4.1)$$

Subsequently, we consider only attributes that are frequent enough to have any significance w.r.t. user's initial selection, i.e., their appearance frequency $f_{a,v}$ should be greater or equal to a predefined threshold θ_L(see line 5). Equ. 4.1 is further used in line 6 to compute the appearance frequency of the remaining candidates in the complete database (by replacing $Lin(r)$ by D). Using the two previous frequencies, we compute the support of each candidate in the provenance w.r.t. the whole database as

$$supp_{a,v}(r) = \left| log_e \left(\frac{f_{a,v}(Lin(r))}{f_{a,v}(D)} \right) \right| \qquad (4.2)$$

Intuitively, if the two considered frequencies significantly differ, the corresponding attribute-value pair is considered as potentially interesting. Thus, interesting attribute value pairs are associated with a high support value and we only keep candidates with $supp_{a,v} \geq \theta_{supp}$ (line 8). The selected attribute-value pairs are added to $RMap$ by calling $mapInsert(RMap, (a, v))$. Essentially, if a does not exist as a key in $RMap$, this function creates a new key-value pair that maps a to a set of values $L_V = \{v\}$. Otherwise, it adds v to the set of values readily existing for key a.

Based on aforementioned steps, we define recommended attribute-value pairs as follows.

Definition 4.1 (Recommended attribute-value pairs) *Given a query Q on a database D, the result $Q(D) = \{r_1, \ldots, r_n\}$ with $n \geq 1$ and a sub-result $r \in Q(D)$ of interest, an attribute-value pair (a, v) is recommended if $f_{a,v}(Lin(r)) \geq \theta_L$ and $supp_{a,v}(r) \geq \theta_{supp}$, where θ_L and θ_{supp} are predefined threshold values.*

The attribute-value pairs grouped together by attribute in $RMap$ result in what we call data recommendations.

Definition 4.2 (Data recommendation) *Data recommendations are pairs (a, L_v) where a is an attribute, $L_v = \{v_1, \ldots, v_n\}$ is a list of values and it holds for all v_i that (a, v_i) is a recommended attribute-value pair.*

Before returning data recommendations, we prune them using data profiling techniques (Abedjan et al. 2015) to avoid redundant data recommendations. More specifically, in lines 10 – 12, we determine whether a functional dependency $a_i \rightarrow a_j$ between two attributes from the computed data recommendations holds. If so, we only retain the data recommendation involving a_i.

4.2 Query reformulation

The data recommendations produced by the above algorithm are input to the query reformulation component that produces, for each (a, L_v), a set of queries corresponding to variations of the original query Q. Each variation reflects a particular operation typical when querying data warehouses. Our system supports variations implementing slice (and dice), drill-down, including the navigation to dimensions not considered in the initial query Q, roll-up, and grouping the original results of Q by measures.

To support users in deciding which of the recommended queries to pursue at a next iteration step, we compute a utility for each recommended query. While many utility functions may be used, our current implementation adopts the Kullback Leiber divergence function (Wasserman 2013) to quantify dissimilarity between the distribution of the grouped attributes of ewvQ' compared to their distribution in D.

$$D_a(Q'(D)||D) = \sum_{i=1}^{n} f_{a,v_i}(Q'(D)) log_e \left(\frac{f_{a,v_i}(Q'(D))}{f_{a,v_i}(D)} \right) \quad (4.3)$$

Given the query reformulations with associated utility scores, users can pick one recommended query Q'. This triggers the recommendation of a suited visualization V' for the query result $Q'(D)$.

4.3 Rendering recommendation

Mapping the query result of a (recommended) query to an appropriate visualization now takes into account both $Q'(D)$ and the previous visualizations users have worked on, as modelled by evolution provenance. We briefly sketch the general procedure, leaving a more thorough formalization, analysis, and optimization for future research.

Given the graph G_X modelling an exploration session and a recommended query Q', which initiates the creation of a new exploration step node in G_X, we consider all evolution provenance on the exploration path, defined as follows.

Definition 4.3 (Exploration path) *Given a node $n \in N$ of the graph G_X, an exploration path P_X is defined as the shortest path from the node n to the root node of G_X that represents the initial seed query Q. We denote the exploration path from the seed query Q to the currently recommended query Q' as $P_X = [Q, Q_1, \ldots, Q_k, Q']$.*

Given an exploration path P_X, we retrieve the evolution provenance associated with each node on the path and each edge connecting nodes on the path using appropriate SQL queries.

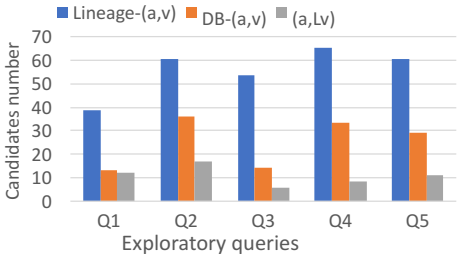

Figure 5: Evolution of candidates number

Using the queries stored in the relation Dataset for each retrieved evolution provenance, we determine the semantic overlap of the data returned by these queries, compared to the data returned by Q'. Intuitively, the higher this semantic overlap, the more similar the queries and hence we want to reuse same or similar visual encodings if possible. We denote the function used to compute the semantic overlap between two queries Q_1 an Q_2 as $sim(Q_1, Q_2)$. To also account for the fact that graphs encountered more recently during an exploration session are more present in user's memory than graphs seen longer ago, we introduce a weight $w_{Q'}(Q, d)$ for a query Q at distance d from the current query Q' that monotonously decreases with an increasing d. For any query $Q_i \in P_X$ on the exploration path yielding Q', we compute a score $sim(Q', Q_i) \times w_{Q'}(Q_i, d)$. For all queries with a score above a threshold θ_V (alternatively, for the top-k queries), we now retrieve the associated evolution provenance about the visualizations associated to those queries.

Then, the most frequent visual encodings used in visualizing information in these queries' results that overlap with information to be visualized for $Q'(D)$ will be used to define a preliminary skeleton for the recommended visualization of $Q'(D)$. In case no prior visualizations can be used or in the initial exploration step (where evolution provenance is empty), we resort to the effectiveness metric defined in Voyager (Wongsuphasawat et al. 2016) to construct the visualization.

Finally, the skeleton for the recommended view is updated following the set of constraints defined in (Qu and Hullman 2016), which are meant to reduce conflict between views. The result of this rendering process is a visualization that is consistent with visualizations previously rendered. The rationale is that recognizing information from previous iterations potentially endorses and facilitates the perception of the next.

5. Implementation and evaluation

We have partially implemented our approach as a web application. The prototype called EVLIN follows the model-view-controller design pattern. It consists of a view layer

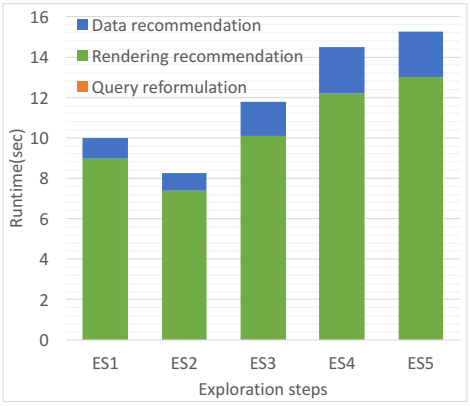

Figure 6: Runtime comparison of recommendation steps

Query	Definition
Q1	SELECT avg(distance), cancellationcode FROM Flights WHERE cancelled=1 GROUP BY cancellationcode
Q2	SELECT count(*),A.state FROM Airports AS A,Flights AS F WHERE A.iata=F.origin AND $deptime \leq 400$ GROUP BY state
Q3	SELECT count(*), C.code FROM Carriers AS C, Tail AS T, Flights AS F WHERE F.uniquecarrier=C.code AND F.tailnum=T.tailnum AND T.manufacturer='BOEING' AND $distance \geq 1500$ GROUP BY C.code
Q4	SELECT sum(carrierdelay),code FROM Carriers AS C, Flights AS F, Airports AS A WHERE A.iata=F.origin AND F.uniquecarrier=C.code AND A.state='CA' GROUP BY code
Q5	SELECT avg(distance), A.state FROM Airports AS A, Flights AS F WHERE A.iata=F.origin AND deptime BETWEEN 2000 AND 2200 GROUP BY A.state

Table 1: Set of queries used in the evaluation

implemented using JSP, Bootstrap 3 and mainly Vega 1[1] for interactive visualizations, a control layer based on struts and implementing the recommendation process using Java 8, and a database backend which is Perm (Glavic and Alonso 2009), an extended version of PostgreSQL that supports provenance information management. The evaluation presented here focuses on the steps of data recommendation and query reformulation, as the implementation of rendering recommendation is still ongoing. An example scenario showing the functionality of EVLIN is available as an online video[2].

Experimental setup. We have performed a set of exploration experiments over a data warehouse dataset about US domestic flights[3]. The measures recorded for each flight include various numerical attributes such as delays, cancellation, distance etc. The dataset contains information about more than two million flights done between 2007 and 2008. In its dimensional tables, we find further information about more than 3300 airports and almost 4500 plane models used by more than 1500 airline companies.

To study the runtime of the different phases of data recommendation and query reformulation, and to evaluate the number of data recommendations (the number of query reformulations is proportional to these), we have defined five queries that conform to Def. 2.1. The proposed queries vary in the used aggregation functions, the complexity of conditions, and join combinations (see Tab. 1).
All experiments were run on a single machine with a 2.2 GHz quad-core Intel processor and 16 GB RAM.

The execution of the data recommendation algorithm is divided in two phases: an offline and an online phase. In an offline phase, we have performed the database frequency computation described in line 3 of Alg. 1 for each attribute present in the flights dataset. We have also identified the set of functional dependencies present in the underlying dataset. During the online phase, provenance computation, data recommendation, query reformulation, and visualization rendering are handled dynamically following the steps described on the left-hand-side of Fig. 1. In our current implementation, Step ⑤ is only partially implemented since we provide currently a table of recommended pairs (a,L_v) without their associated interestingness scores. The user selects such a pair to get its associated recommended visualization.

Evaluation results. Our first experiment considers the evolution of the number of candidate recommendations through the three major steps of our data recommendation process.

More specifically, we report in Fig. 5 three measurements for each query: (1) the number of candidate attribute-value pairs that are candidates after considering the provenance of r (i.e., candidates remaining at line 6 of Alg. 1), (2) the number of recommended attribute-value pairs (cf. Def. 4.1) processed at line 9 of Alg. 1, and (3) the final number of data recommendations output by the algorithm. We set $\theta_L = 0.1$ and $\theta_{supp} = 0.7$ which proved to be practical for the data set we studied in this experiment. However, further work is required to provide a more generic setting mechanism for these threshholds.

The results indicate that each candidate pruning step, as well as the grouping of attribute values by attribute, is effective and allows to reduce the number of recommendations to be processed by the user to a manageable number. Also, the returned recommendations are less redundant and more concise, potentially having a positive influence on the user satisfaction.

The next experiment studies the runtime of *data recommendation*, *query reformulation* and *rendering recommendation* for the same five queries used previously. Each query is used as a seed query, first triggering rendering recommendation. A user then randomly selects $r \in Q(D)$, an interaction that triggers data recommendation. Query reformulation is finally initiated once a user randomly chooses a single data recommendation. A data recommendation will result in three queries describing a zoom in, drill down, and navigation to a new dimension in the data warehouse.

[1] https://vega.github.io/vega/

[2] https://www.youtube.com/watch?v=GgnUCQO_2DU

[3] https://stat-computing.org/dataexpo/2009/

System	input query	recom. query	recom. vis.
YmalDB (Drosou and Pitoura 2013)	SPJ	SPJ	table
SeeDB (Vartak et al. 2015)	cube	sub-cube	bar chart
REACT (Milo and Somech 2016)	cube	OLAP queries	table
Voyager (Wongsuphasawat et al. 2016)	PA	Changed SELECT-clause	diverse

Table 2: Summary of related work

Fig. 6 shows the results, averaging runtimes of each stage over 15 runs of the experiment. We first observe that the time needed for query reformulation runtime is negligible compared to the other two steps. Next, we see that rendering recommendation takes the most time. This is mainly due to the costly serialization of recommended query results and visualization skeletons in JSON format that is required in order to comply with the Vega specification used in our prototype to render results. Finally, the computation time of data recommendations remains acceptable (between 0.5 and 3 seconds).

6. Related Work

As mentioned in the introduction, visual data exploration systems either rely on query recommendation or visualization recommendation. Opposed to EVLIN that considers both types of recommendations, most data and visualization recommendation techniques work independently from one another, meaning that the result of query recommendation, i.e., $Q'(D)$ has no impact on the visualization recommendation process, and vice versa.

Tab. 2 provides a quick overview of works most closely related to ours. For each approach, it describes (1) the supported types of input queries (e.g., select-project-join (SPJ) queries, project-aggregate (PA) queries, or cube queries corresponding to SPJA queries), (2) the type of recommended output queries, and (3) the type of recommended or used visualization. We observe that while query recommendation systems such as YmalDB (Drosou and Pitoura 2013), SeeDB (Vartak et al. 2015), or REACT (Milo and Somech 2016) allow for expressive data exploration by supporting (mostly) the full range of typical OLAP queries, they do not offer any visualization recommendation and simply use predefined visualization (table or bar chart). Opposed to that, visualization recommendation systems such as Voyager (Wongsuphasawat et al. 2016) do not yet offer or have only very limited support for query recommendation. Our system aims at bridging the gap between these two approaches. In addition, to the best of our knowledge, EVLIN is the first system that considers data provenance and evolution provenance during its recommendation processes.

7. Summary and Future Work

This paper presented our visual data exploration system as well as the provenance-based recommendation techniques that we use to provide both query recommendations and visualization recommendations in an integrated way. In particular, we have presented how we model evolution provenance, meta-data that is currently used to influence visualization recommendation. Next, we described the different steps leading to query recommendations. We further sketched our ideas for visualization recommendation using evolution provenance. We have implemented a system prototype and presented some preliminary experiments.

Next steps include a more thorough definition of visualization recommendation as well as optimization techniques to improve runtime performance. Also, a more thorough experimental evaluation is needed to better understand the overall system behavior.

Acknowledgements. The authors thank the German Research Foundation (DFG) for supporting the project D03 of SFB/Transregio 161.

References

Z. Abedjan, L. Golab, and F. Naumann. Profiling relational data: a survey. *VLDB Journal*, 2015.

I. Altintas, O. Barney, and E. Jaeger-Frank. Provenance collection support in the kepler scientific workflow system. In *IPAW*, 2006.

M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. In *TVCG*, 2011.

S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, T. Vo, and H. T. Silva. VisTrails : Visualization meets Data Management. In *SIGMOD*, 2006.

J. Cheney, L. Chiticariu, and W. C. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1 (4), 2009.

M. Drosou and E. Pitoura. Ymaldb: Exploring relational databases via result-driven recommendations. *VLDB Journal*, 2013.

B. Glavic and G. Alonso. The perm provenance management system in action. In *SIGMOD*, 2009.

S. Idreos, O. Papaemmanouil, and S. Chaudhuri. Overview of Data Exploration Techniques. *SIGMOD*, 2015.

T. Milo and A. Somech. React: Context-sensitive recommendations for data analysis. In *SIGMOD*, 2016.

Z. Qu and J. Hullman. Evaluating visualization sets: Trade-offs between local effectiveness and global consistency. In *BELIV*, 2016.

A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. In *TVCG*, 2017.

M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. SEE DB : Efficient Data-Driven Visualization Recommendations to Support Visual Analytics. *PVLDB*, 2015.

L. Wasserman. *All of statistics: a concise course in statistical inference.* 2013.

K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory Analysis via Faceted Browsing of Visualization Recommendations. *TVCG*, 2016.

E. Wu, F. Psallidas, Z. Miao, H. Zhang, and L. Rettig. Combining design and performance in a data visualization management system. In *CIDR*, 2017.